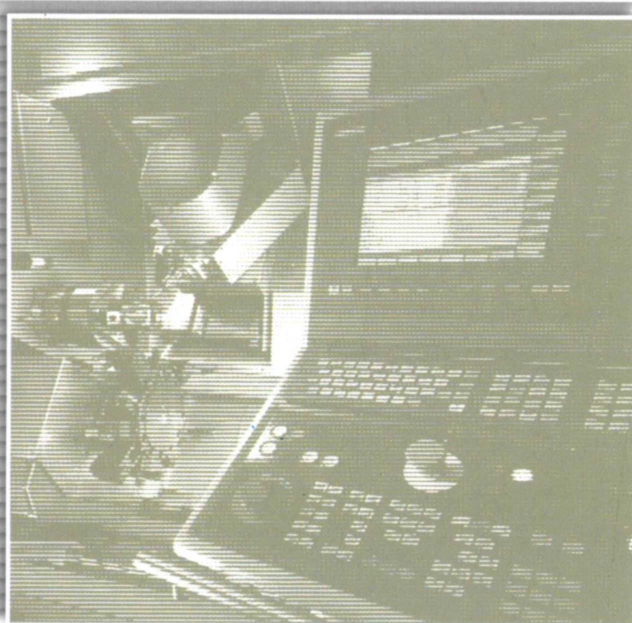
多轴Open-CNC制造系统基础理论与技术

陈良骥　马雄波　著





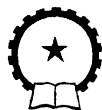
机械工业出版社
CHINA MACHINE PRESS

VISIT…

LANZAROTE Caliente.COM

多轴 Open-CNC 制造系统基础理论与技术

陈良骥　马雄波　著



机 械 工 业 出 版 社

本书系统、全面地介绍了开放式数控体系结构的基本原理及实现多轴CNC制造系统的基础理论与方法，研究了相应的多轴Open-CNC制造系统的实现技术和核心算法。主要内容包括多轴CNC系统的开放技术及其体系结构、多轴Open-CNC系统代码编译解释器工作原理、多轴Soft-PLC可编程序逻辑控制器实现机理、多轴加工轨迹生成理论及其精密插值方法、制造系统的搭建与实现技术等。

本书适合于国内各相关高等院校、科研院所、民用军用工业企业中从事多轴数字化制造技术的师生和研究人员参考和使用。

图书在版编目（CIP）数据

多轴Open-CNC制造系统基础理论与技术/陈良骥，马雄波著.—北京：机械工业出版社，2013.7

ISBN 978-7-111-42574-8

Ⅰ.①多… Ⅱ.①陈… ②马… Ⅲ.①数控机床-程序设计-高等学校-教材 Ⅳ.①TG659

中国版本图书馆CIP数据核字（2013）第104867号

机械工业出版社（北京市百万庄大街22号 邮政编码100037）
策划编辑：沈 红 责任编辑：沈 红
版式设计：常天培 责任校对：张 媛
责任印制：乔 宇
北京机工印刷厂印刷（三河市南杨庄园丰装订厂装订）
2013年7月第1版第1次印刷
169mm×239mm · 7.5印张 · 140千字
0 001—2 000册
标准书号：ISBN 978-7-111-42574-8
定价：29.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务	网络服务
社服务中心：（010）88361066	教材网：http://www.cmpedu.com
销售一部：（010）68326294	机工官网：http://www.cmpbook.com
销售二部：（010）88379649	机工官博：http://weibo.com/cmp1952
读者购书热线：（010）88379203	**封面无防伪标均为盗版**

前　言

开放式数字控制（Open-CNC）的设计思想形成于20世纪80年代，伴随着计算机系统技术的快速发展，现代CNC制造系统正朝着开放性（openess）、软件化（software）的方向发展。同时soft型可编程序逻辑控制器（PLC）技术的诞生及可编程序逻辑控制器国际标准（IEC61131-3）的推行，也促使了在Open-CNC制造系统中利用软件技术来实现对PLC硬件单元的逻辑控制成为可能。

本书基于工业控制机（IPC）平台、采用通用的Windows操作系统及RTX实时扩展，以及遵循SERCOS协议的SoftSERCANS通信卡，建立了可搭建开放式数字控制器软硬件平台的多轴Open-CNC制造系统基础理论方法与实现技术。利用上述基本理论，可在该平台下实现多轴Open-CNC控制器与数字伺服电动机驱动器及I/O接口设备之间的高速串行通信，从而可实现较好的软硬件无关性，并有效地提高了系统开放性，使整个制造系统顺利成为运行在通用IPC机上的一个应用软件。研究基于通用软硬件平台的多轴Open-CNC制造系统对于当代航空航天、汽车制造等先进数字化切削制造领域的高速、高效、高精密加工制造具有极大地促进作用、重要的科学意义和实际应用价值。

本书内容翔实、新颖，对现行多轴数字化制造领域中存在的诸多问题和难点分别进行了深入浅出的分析，并给出了相应的基础理论和相关的解决方案和技术路线。本书的主要内容及实现机理有：参照OMAC API协议、采用有限状态机模型及应用SERCOS技术；采用编译方式进行加工代码解释，将代码信息翻译成可供插值模块执行的运动点坐标、速度及PLC开关量等，通过共享内存传递给插值模块以控制机床运动；参照IEC61131-3，分析并确定soft型PLC开发原理与基本方法，实现了梯形图编辑器和指令表编辑器，以及由梯形图到指令表的转换功能；介绍了双NURBS样条插值原理和技术。本书中所涉及的多轴Open-CNC制造系统相关基础理论和技术对数字化精密制造领域具有较好的理论指导意义和实际推广应用价值。

本书由郑州航空工业管理学院陈良骥和淮海工学院马雄波合著而成。陈良骥撰写了第1章和第5章的内容，马雄波撰写了第2章、第3章、第4章和第6章的内容。

书中所涉及的大部分内容在研究阶段得到了国家自然科学基金项目（51275485）的大力支持，同时获得了河南省高校科技创新人才支持计划（13HAST17036）的资助及郑州航空工业管理学院、淮海工学院的鼎力协助，作者在此一并表示诚挚的感谢。

作　者

目　　录

前言
第 1 章　绪论 …… 1
1.1　Open-CNC 理论产生背景 …… 1
1.2　Open-CNC 理论相关现状分析 …… 1
1.2.1　国外现状分析 …… 2
1.2.2　国内现状分析 …… 5
1.3　Soft-PLC 理论现状分析及意义 …… 7
1.4　Open-CNC 制造系统发展趋势分析 …… 9
1.5　研究项目来源 …… 9
1.6　本书目的及其理论意义 …… 10
1.7　主要内容安排 …… 11
第 2 章　Open-CNC 基础理论与系统架构 …… 12
2.1　接口标准 …… 12
2.2　系统实时性实现原理 …… 14
2.2.1　何谓实时 …… 15
2.2.2　实时与非实时系统的比较 …… 15
2.2.3　实时系统间的比较 …… 15
2.2.4　实时系统的确定 …… 17
2.2.5　RTX 实时系统的利用 …… 17
2.3　OMAC 工程及 OMAC API …… 19
2.3.1　动态链接库（DLL）技术 …… 19
2.3.2　客户机/服务器通信模式 …… 21
2.4　有限状态机在 CNC 中的实现机制 …… 22
2.5　SERCOS 接口原理与实现方法 …… 24
2.5.1　SERCOS 接口简介 …… 24
2.5.2　SERCANS 软件功能 …… 25
2.6　Open-CNC 制造系统的架构 …… 26
2.6.1　系统架构 …… 26
2.6.2　系统组成 …… 26
2.7　小结 …… 28
第 3 章　Open-CNC 制造系统的代码解译机理 …… 29
3.1　概述 …… 29
3.2　加工代码解释器结构设计 …… 29

3.3 加工代码的段格式 …… 30
3.4 加工代码的解释实现技术 …… 30
3.4.1 译码数据处理过程 …… 30
3.4.2 读入加工代码 …… 32
3.4.3 解释加工代码 …… 33
3.5 出错处理机制 …… 34
3.5.1 常见出错分析 …… 34
3.5.2 错误处理方法 …… 34
3.6 任务生成指令及其功能 …… 35
3.6.1 指令分组原则 …… 35
3.6.2 固定循环指令 …… 37
3.6.3 坐标系建立方式 …… 38
3.6.4 系统参数设定 …… 38
3.6.5 可扩展宏的定义 …… 39
3.6.6 运算符与函数定义 …… 40
3.6.7 注释代码 …… 40
3.6.8 子程序调用 …… 40
3.7 刀具补偿机理 …… 41
3.7.1 概述 …… 41
3.7.2 数据存储管理模式 …… 41
3.7.3 刀具补偿实现机理 …… 42
3.8 小结 …… 43
第 4 章 Soft-PLC 技术原理及实现 …… 45
4.1 概述 …… 45
4.1.1 以软件实现传统 PLC 功能 …… 45
4.1.2 IEC61131-3 软件系统 …… 46
4.2 Soft-PLC 系统结构 …… 46
4.3 Soft-PLC 编程开发系统设计 …… 47
4.3.1 编程开发系统中编辑器的实现 …… 48
4.3.2 开发系统中编译器的实现 …… 48
4.4 Soft-PLC 运行系统设计 …… 53
4.4.1 系统任务划分 …… 54
4.4.2 Soft-PLC 基本扫描任务 …… 56
4.4.3 任务调度 …… 56
4.4.4 Soft-PLC 数据结构 …… 58
4.5 人机控制 …… 59
4.5.1 HMI 与 Soft-PLC 的通信 …… 59
4.5.2 HMI 设计 …… 62

4.6　小结 …… 62
第5章　Open-CNC 制造系统多轴精密插值机理 …… 64
5.1　概述 …… 64
5.2　数据流处理 …… 65
5.3　线性插值机理 …… 67
5.3.1　制造系统插值过程设计 …… 68
5.3.2　进给加减速实现方法 …… 69
5.4　多轴样条精密插值的实现机理 …… 72
5.4.1　样条曲线表示方法 …… 72
5.4.2　双 NURBS 曲线插值格式 …… 73
5.4.3　加工轨迹的 NURBS 拟合方法 …… 74
5.4.4　双 NURBS 曲线数据流处理 …… 74
5.4.5　加减速处理 …… 79
5.5　小结 …… 86
第6章　多轴 Open-CNC 制造系统的实现 …… 87
6.1　设备联机调试及集成机制 …… 87
6.2　制造系统的搭建 …… 90
6.2.1　三轴系统 …… 90
6.2.2　五轴系统 …… 93
6.2.3　系统的开放性 …… 97
6.3　Soft-PLC 系统实验 …… 97
6.3.1　开发系统 …… 97
6.3.2　运行系统 …… 99
6.4　小结 …… 100
结论 …… 101
参考文献 …… 103

第 1 章　绪　　论

1.1　Open-CNC 理论产生背景

自 1952 年世界上第一台数控机床诞生以来，数控技术伴随现代科学技术的不断发展，已日趋完善。但随着科技和生产的发展，数控技术面临着众多新的挑战，如生产中不断出现了新的加工需求，这就要求计算机数控（Computer Numerical Control，CNC）制造系统应具有迅速、高效、经济地面向用户的模块化结构；要求数控系统改变传统的封闭式结构，降低机床厂和用户对数控系统厂商所提供控制模式和策略的依赖性，适应未来生产“面向任务和订单”的生产模式和满足“可重构制造系统”的需求。

这样一来，CNC 数字化制造系统必须具有可重构、可扩展、可移植和可伸缩等特性，并允许用户在集成传感器和加工监控系统的基础上方便地实现“智能控制”。要达到这一目的，最有效的途径就是实现数控系统的开放性（openness）。在开放的体系构架下，用户可以自由地选择数控系统的各个构成要素，并按照规范的、简便的方法组成新的 CNC 系统，以实现生产率最大化和质量最优化。数控系统由封闭式控制模式向开放式控制模式发展，是当前数控技术发展的必然的、重要的趋势。

开放式数字控制（Open-CNC）制造系统的研究从 20 世纪 80 年代末开始，到现在还没有形成统一的标准，因而也还没有一个统一的、明确的定义。IEEE 是这样定义开放系统的[1]：“具有下列特性的系统可称为开放系统：符合系统规范的应用可运行在多个销售商的不同平台上，可与其他的系统应用互操作，并且具有一致风格的用户交互界面。”

针对开放式数控系统的应用需求，一般认为，开放式数控系统具有以下基本特征，即可互换性（interchangeability）、可移植性（portability）、可伸缩性（scalability）、互操作性（interoperability）、可扩展性（expandability），这也是衡量数控系统开放程度的准则[2-4]。

1.2　Open-CNC 理论相关现状分析

开放式数控系统的核心是具有开放性的控制器。它必须站在用户的立场，方便、快捷、稳定、可靠和全面开放。随着制造业技术的发展，对数控机床的

柔性和通用性提出了更高的要求。因此，各国相继对控制器的开放性展开了研究。

1.2.1 国外现状分析

最早关于开放式体系结构控制器的研究源于美国。1981 年，美国发起了一个叫下一代控制器的计划，简称 NGC（Next Generation Workstation/Machine Controller）[5]，以实现基于互操作和分级式软件模块的“开放体系结构标准规范”SOSAS（开放体系结构标准规范）。NGC 包括两个主要的部分：制定开放式体系结构标准规范（SOSAS）和建立三个 NGC 的参考模型[6,7]。SOSAS 最重要的部分是，允许单独的设计者去开发具有互换性和相互协作的 NGC 单元，并包括 CNC 兼容产品的所有功能，为以后开发的产品提供了运行的基础[8,9]。由于经济的原因，该计划在 1991 年中断。自 1992 年以来，欧盟、美国、日本等制造业发达国家相继开展了大规模的开放式数控系统的研究项目。主要有：

（1）欧洲 OSACA 项目　1991 年 11 月，ESPRIT Ⅱ 开始的一项整个欧洲的控制系统计划 OSACA，其目标是研制出自动化控制系统中的开放式体系结构。OSACA 计划于 1992 年 5 月启动，在第一阶段（1992 年 5 月到 1994 年 4 月）内，给出了独立于厂家的开放式控制系统的基本规范。规范中详细说明了系统平台的基本功能规范，平台的系统软件应主要由操作系统、通信系统和配置系统组成，还拟订了独立于公司的参考结构，以用于数控、机器人、可编程序控制器和单元控制。在第二个阶段（1994 年 5 月到 1996 年 4 月），由所有参加成员为此平台联合开发的软件建立了一个公用系统软件库，用于测试、验证和扩展，还开发了第一批示范软件模块。这之后的工作主要是将 OSACA 建立为开放式控制系统的工业标准[10]。该计划结束于 1998 年 7 月，历经 66 个月，投入 96 人年，1230 万欧洲货币单位。其主要目标是建立一个开放性的、厂商无关的控制器体系结构[11]。

OSACA 是这样定义开放系统的：“开放的控制系统是由一组逻辑独立的功能部件（component）组成，这些部件之间、部件和其实现平台之间具有开放、标准的接口定义。一个完整的控制系统可由来自于多个厂商的部件协作而成，这个控制系统可以运行在多种平台上，并且具有与其他自动化系统一致的人机接口”。

可以看出，“功能部件”是 OSACA 开放系统的核心，为了有效地描述这些“功能部件”的属性和集成关系，OSACA 全面采用面向对象的分析方法。OSACA 借鉴 ISO 的 OSI 参考模型，通过对现有控制系统的详细分析，提出一个“分层的系统平台 + 结构功能单元”的结构。OSACA 提出的开放式数控系统模型如图 1-1 所示。从其体系结构来看，OSACA 控制系统分为两个部分：应用软

件和系统平台。应用软件由一组被称为"体系结构对象 AO（Architecture Object)"的功能模块组成，这些 AO 具有很强的功能独立性，也是 OSACA 体系结构的灵魂；应用软件可运行在多种 OSACA 兼容的平台上。系统平台由系统硬件和系统软件组成；系统硬件主要指控制系统的各种电子部件，如处理机、I/O 模块等；系统软件包括操作系统、通信系统及附加的各种设备的驱动程序。AO 之间的通信是采用面向对象的信息模型（通信对象）来实现的，以客户/服务器模式为基础。通信对象包括变量对象、进程对象、事物对象[12,13]。

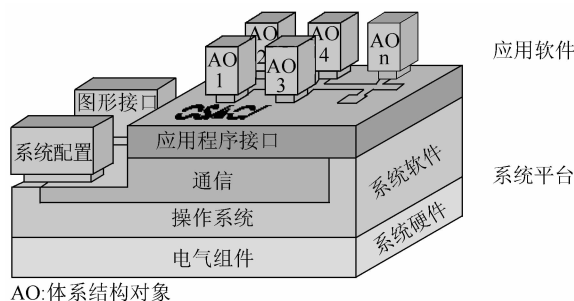


图 1-1　OSACA 体系结构[10]

（2）日本 OSEC 项目　1994 年 12 月，日本的东芝机器公司、丰田机器厂和 Mazak 公司三家机床制造商与日本 IBM、三菱电子及 SML 系统公司联合发起了 OSEC（Open System Environment for Controller）项目。该项目共分三个阶段，结束于 1999 年 3 月。OSEC 的目标是为下一代工厂自动化设备控制器建立一个开放性的体系结构规范，并开发相应的接口标准[14]。

OSEC 采用功能组定义了开放式数控系统的体系结构，如图 1-2 所示。主要包含 4 个功能组，即运动生成、资源管理、加工控制、设备控制。每个功能组由若干个功能模块组成，每个功能模块是一个对象形式的软件元件，并封装了 OSEC API（Application Program Interface）形式的对象消息作为接口协议。功能模块之间不形成层次关系，通过 OSEC API 形式的消息连接起来进行通信和协作。OSEC 体系结构只定义了功能模块的服务内容和消息接口协议，由控制器厂商进行功能模块的个

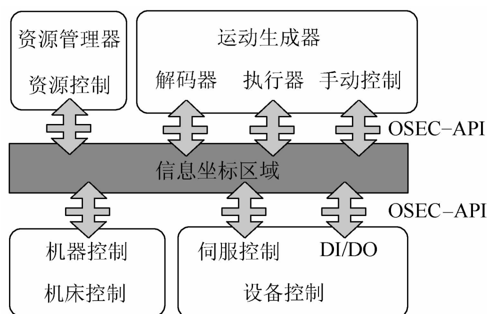


图 1-2　OSEC 体系结构[14]

性化实现，符合接口协议的功能模块在 OSEC 控制器中可以“即插即用”。

（3）美国 OMAC 项目 1994 年底，美国三大汽车公司通用（GM）、福特（Ford）和克莱斯勒（Chrysler）首次提出了 OMAC（Open Modular Architecture Controllers），即开放式、模块化体系结构控制器的概念[15]，称开放化就是允许把流行的硬件和软件集成为控制器的基础结构。它支持一个事实上的标准环境。1997 年 2 月，OMAC 用户组成立，对 OMAC 感兴趣的组织随时可以加入，并参与相关的技术开发。OMAC 的主要目标是明确用户对于开放体系结构控制器的应用需求，开发一种满足这种需求的公共 API，为开放式控制器技术开发、实现和商品化中的各种问题提供共同的解决方案[16,17]。

OMAC 采用组件技术实现即插即用的模块化，采用接口类的形式定义 API。OMAC 定义了各种不同“尺寸”和“类型”的可重用即插即用“组件”，这是广义的组件，是指构成应用程序的软件片，实际上包括 COM（Component Object Model）组件、模块、任务。每一个“组件”都具有各自的有限状态机实现其特定功能。模块是指包含组件的容器。任务是指封装可编程序功能行为的组件，包括一系列待完成的步骤，如启动、停止、暂停、恢复。当控制器工作时，任务可以多次运行。在分布式通信情况下，基于组件的技术采用代理存根的方式处理跨进程的方法调用。

OMAC 没有定义一个固定的参考结构，但是定义了用于建造不同类型控制器的模块集[3,17,18]。OMAC API 的目标是使控制系统销售商提供标准组件，机床供应商把这些组件配置到机床控制系统中，然后把这种控制系统和机床交付给最终用户。OMAC API 应用涉及的范围从单轴控制器到多轴控制器。OMAC 工作组努力使 OMAC API 规范成为被业界承认的标准。OMAC API 使用基于组件的方法获得即插即用的特性，使用界面类指定 API。OMAC API 包含不同粒度和类型的即插即用的元件——组件、模块和任务。组件是一种可复用的软件，它是应用程序的一个建造块；模块是组件的容器；任务是用于封装可编程序功能行为的组件，功能行为由一系列步骤组成。这些步骤包括启动、停止、重启、暂停、继续，在控制器运行时这些步骤可能循环多次。OMAC API 使用微软的 COM 技术进行集成，这样数控系统开发商可以集中精力进行专用软件的开发。使用 COM 的主要问题是硬实时性能欠佳，这个问题通过对 Windows 操作系统进行实时扩展的方法来解决。

（4）国外其他机构的研究情况 除以上影响力最大的三个研究工程外，国际上还有许多其他的机构和大学在从事开放体系结构控制器的研究，并建立了一些开放式数控系统的实例[3,19-22]。如美国密歇根大学（University of Michigan）在开放体系结构控制器领域成果显著[23,24]，其主要目标是实现机床控制系统的开放性。研究内容包括：基于 FSM 的机床控制设计[25,26]，基于 Windows 的 HMI

的开发，可重构机床的控制结构设计[27,28]，且在加工力分析、颤动分析与监测、刀具和工件接触监控等方面也进行了大量的研究工作[29-31]。他们也在数控软组件模块化方面进行了研究[32,33]。普渡大学（Purdue University）智能制造实验室开展了基于PC的开放式体系结构控制器、加工仿真建模和实现、加工过程的多自变量自适应控制、常规切削力自适应控制等研究[34]。澳大利亚研究人员则利用DSP技术开发了位置闭环控制的伺服驱动系统[35]。由于涉及重新编制DSP驱动程序的工作，因此重构系统和扩展功能仍然相对复杂。加拿大不列颠哥伦比亚大学（University of British Columbia，Canada）制造自动化实验室也开展了大量的研究，如开发了一个基于DSP的智能加工模块（Intelligent Machining Module，IMM），即由一组用户可扩展的函数库组成。研究人员在IMM平台基础上可以快速实现智能加工工艺算法，可以将刀具磨损监测、热变形补偿等以传感器技术为基础的智能控制算法迅速集成到数控系统中；IMM实际上是一个以开放体系结构为基础的智能数控开发平台[36,37]。德国斯图加特大学（University of Stuttgart）具有长期从事开放模块化控制系统设计的历史[38,39]。20世纪80年代，MPST项目以并行总线连接硬件模块建立了模块化的结构控制器。后来，他们开始进行基于软件的模块化控制器体系结构的研究。以OSACA准则为基础建立了开放式控制器，可应用于多轴车、铣、并联机构及其他特殊用途的机床（如电火花、纺织机械等）。另外还有一些研究机构在开放式数控方面的研究工作具有较大的影响，如日本东京大学[40]、加州大学伯克利分校（University of California，Berkeley）[41,42]等。

1.2.2 国内现状分析

我国在“八五”和“九五”期间，重点投资北京机电数控集团、航天数控集团和华中数控集团，开发了以PC机为基础的中华I型、航天I型、华中I型和蓝天I型4种基本系统。这些系统尽管基本建立在PC平台上，但它们的系统结构仍属于封闭的。

从20世纪90年代起，我国很多科研机构和大学纷纷投入力量进行了开放式数控系统方面的研究和开发。如华中理工大学周祖德教授等提出了一种基于软件芯片的开放式数控系统的实现模式[43-46]，它实际上是采用面向对象的编程技术，借鉴了欧盟的OSACA研究方法。通过对数控系统的功能划分和接口规范化，运用面向对象的机制，把数控系统的功能进行抽象并进行封装，将数控软件设计成具有稳定、通用接口可以重用的软件芯片，且每一个软件芯片完成数控系统的一个独立模块的功能，如插值功能就由插值芯片来完成。通过建立一个数控系统软件芯片集成开发环境——数控系统软件芯片库，来对软件芯片进行管理，用户可以对软件芯片库进行检索、浏览以及维护，还可以添加自己制

作的软件芯片。用户在开发新的数控系统时，只需从这些芯片软件库中取出所需的芯片进行组合即可。北京航空航天大学的陈五一教授等提出了基于 RT-Linux 的开放式数控系统的概念，并在 RT-Linux 的系统平台上，实现了基于组件的开放结构的数控系统的原型系统[47-50]。该系统以任务模块作为系统功能单元，以虚拟模块系统实现数控系统功能单元间的信息交换与同步，以配置系统负责系统的集成。它支持系统在启动时重构，用户只要离线编写配置脚本书件，由配置系统负责系统的重构，即可实现系统的良好开放性。北京航空航天大学的郇极教授等提出了基于 SERCOS 技术的开放式数控系统概念，并开发了基于 SERCOS 的开放式数控系统 CH-2010/S[51]。由于采用 SERCOS 技术，CH-2010/S 的硬件结构十分简单，但是却具有很强的控制功能和灵活性，且可以控制 1～50 个伺服电动机和 40～320 个 I/O 点，并快速完成系统的扩充、裁减或部件替换，进而在同一硬件平台上，可以开发多种数控系统。CNC 与伺服之间采用光缆连接，大大减少了电缆布线。可提供丰富的诊断信息，便于系统安装和维护，提高了系统可靠性。光缆连接也特别适用于大型机床所需要的远距离控制功能。上海交通大学的研究人员研究了开放体系结构数控系统内核的重构方式，并提出了一种新的基于行为的动态建模方法，如图 1-3 所示。利用这种方法，分析了开放系统模型中应用对象的生命周期和相互作用关系。他们认为开放体系结构数控系统内核应由 MCM（Motion Control Management，运动控制任务管理）、MC（Motion Control）、AC（Axis Control，轴控制）、SC（Spindle Control，主轴控制）等任务管理区域组成。因此系统内核可以设计成为软件模式的多通道结构，且每个通道的轴数和运动轴类型由配置系统设定[52]。另外，还出现了基于数字现场总线的开放式数控系统[53-56]、基于运动控制卡的 PC＋NC 数控系统[57]、基于 Windows 的开放式数控系统[51]、基于 COM/DCOM 组件的开放式数控系统[58]，以及智能数字控制器[53]等。此外，成型的产品包括中科院沈阳计算所/高档数控国家工程研究中心的 SS-8540、机械部机床研究所的中华 I 型、华中理工大学的 PCNC 数控系统。清华大学[59]、天津大学[60]等也进行了相关方面的研究。

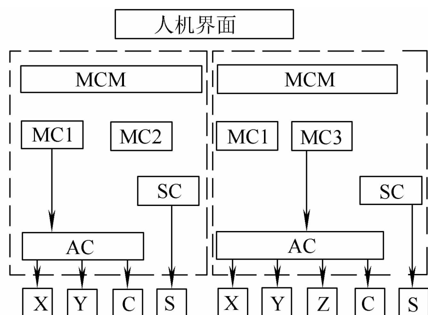


图 1-3 多通道结构的 CNC 内核[52]

由于受硬件研发及生产能力的制约，国内的主要数控系统都在朝 PC 化方向发展，但 PC 化并不等于开放。各系统所采用的体系结构并不一致，仍是自成体系，相互之间缺乏兼容性和互换性，而且对体系结构的阐述都只限于具体实现

层，没有提高到理论的、抽象的层次上来，因而各系统软硬件不具备可移植性和互操作性，也更谈不上开放。虽然国内在开放式数控系统研究方面有一定的广度，但大多局限于概念上的研究及理论上的分析，总体研究水平与国外相比尚有较大差距。我国从事开放式数控系统起步较晚，且规模还无法与国外相比，即多限于某一体系结构的具体实现。

1.3　Soft-PLC 理论现状分析及意义

在 Open-CNC 制造系统中，Soft 型可编程序逻辑控制器（Soft-PLC）是作为开放式控制器体系结构中完成离散事件控制的模块来研究的，其跟随着开放式软数控系统的发展。

（1）国外发展动态　国外在这方面的研究相对较早，且主要还是围绕在开放式控制系统领域研究最具代表性的三个国际工程进行的。美国 OMAC 项目规划的控制器结构就是将离散逻辑作为系统中一个标准的模块，与轴组模块、任务生成器处于同一层次，由系统任务协调器统一调度。目前，作为较成熟的开放式数控系统产品推出市场的有美国 MDSI 公司的 OpenCNC、德国 Power Automation 公司的 PA8000 NT 及美国 Soft SERVO 公司的 ServoWorks。

其中 ServoWorks 是基于 Windows NT/2000/XP 和 Linux 全软件开放式数控系统，由美国麻省理工学院的专家创办的 Soft SERVO 公司推出的基于计算机平台真正开放的全软件运动控制产品。ServoWorks 实时控制软件主要包含三个模块：运动引擎、运动解释器和 PLC 引擎。实时 Soft-PLC 模块与 ServoWorks 运动引擎紧密结合并监控运动状态，用于机床运动、逻辑控制，其执行唯一 API 功能，可执行与 FANUC 数控系统兼容的梯形图逻辑，以及提供 38 种功能指令及 12 个基本指令[61]。

（2）国内发展动态　国内有些院校、科研机构在这方面也展开了有针对性的基础研究工作，并取得了不错的成果。但仍处在实验项目阶段，尚未开发出比较成熟的产品。

国家数控系统工程与研究中心开展了开放式数控系统中的 Soft-PLC 技术的研究，提出一种在基于开放式数控系统的嵌入式 Soft-PLC 实现方法。结合本身数控系统的构造思想，给出了一个实现 Soft-PLC 的上下位机模式。并对 PLC 用户开发环境、梯形图和指令表编程、编程语言间的相互转换及 PLC 执行等软件部分进行了设计和开发的研究[62]。

清华大学制造工程研究所开发了一种基于 RT Linux 实时操作系统的 Soft-PLC，其既可以作为一个独立的应用软件和不同的 CNC 系统协调工作，也可以嵌入 THHP-Ⅱ型数控系统内部在同一计算机上运行。其实时性和可靠性已经在

国产 THM5660H 和 TH5456 立式加工中心上得到验证[63]。

此外还有北京工业大学、四川大学、华中科技大学也在这方面做了一些基础研究工作[64-67]。

(3) Soft-PLC 理论研究意义 随着计算机软硬件技术的飞速发展，计算机及其外围设备的性能价格比、平均无故障率等技术指标均优于专用 PLC 控制器，在计算机上以软件实现 PLC 成为新的发展热点，欧美等国家都将 Soft-PLC 控制器作为一个重点投资对象进行研究，目前已向市场投入一些成熟产品。而 Soft-PLC 概念在 1996 年以后才被介绍到国内，所以在这方面的技术尚不成熟。不过随着可编程序控制器国际标准 IEC61131-3 在国内的大力推行，为实现开放式控制器系统提供了理论依据，也促进了 Soft-PLC/PC 的集成控制作为新兴产业的发展。

自 20 世纪 80 年代以来，开放体系结构的控制器成为数控领域的发展主流和研究热点，更是新一代数控的关键技术。开放式数控系统的发展历程中，软件型开放式数控系统成为一种最新开放体系结构的数控系统，它提供给用户最大的选择和灵活性。CNC 软件全部装在计算机中，而硬件部分仅是计算机与伺服驱动和外部 I/O 之间的标准化通用接口。软件智能替代复杂的硬件，正在成为当代数控系统发展的重要趋势。

研究 Soft-PLC 可以打破以往各 PLC 生产厂家产品互不兼容的局限性，缩短产品开发周期，并使 PLC 性能价格比提高，以及带来巨大的经济效益。PLC 的发展在 IEC1131 标准推出以后，尤其是其中关于 PLC 编程方面的 IEC1131-3 标准推出后，变得十分迅速。目前，Soft-PLC 已经在国外形成了规模性的研发机制，但国内 Soft-PLC 形成产品化生产的企业还不多。我国在控制领域依然落后于西方国家，且这些研究还没有得到广泛的推广。国内对 IEC1131 标准的介绍也通常是一个综述性的表述，没有一个探讨性的说明。

将 Soft-PLC 技术和 IEC61131-3 标准引入开放式软件数控系统的研究中，以子程序或控制模块软件形式来实现控制机床逻辑的 PLC 单元，可以充分地利用计算机软件资源，进一步提高开放式数控系统的灵活性、开放性；同时是研究 Soft-PLC 技术及其应用配合很好的机会。目前，国际上对 Soft-PLC 研究的技术没有完全成熟，我国应该抓住这个契机，在国际数控设备激烈竞争的环境中，开发出具有自主知识产权的国际一流的高性能数控软件，推动我国数控技术的发展，缩短此行业与发达国家的差距，发展我国制造业。因此，有必要把握时机，紧跟前沿技术的进展，研究且解决关键技术，为国产开放式数控系统和 Soft-PLC 产品的研制开发打下坚实的基础。

1.4　Open-CNC 制造系统发展趋势分析

数控技术是数控机床的关键技术，是柔性制造系统（FMS）、计算机集成制造系统（CIMS）和工厂自动化（FA）的技术基础。随着微电子技术和计算机技术的迅速发展，特别是 IPC 技术的高速发展，使开放式数控成为数控技术的发展方向。

从当前国内外的研究情况来看，实现开放式数控系统的方式有三种：第一种是将 IPC 板卡嵌入到专用数控系统中；第二种是将 NC 板卡嵌入到通用 IPC 机中；第三种是采用完全基于通用 IPC 机的软件化数控方式。

将 IPC 板卡嵌入到专用数控系统中，既可以保留原有的专用数控系统，又可以开放数控系统的人机界面，因而被专用数控系统制造商（如 SIEMENS 和 FANUC 等）所广泛采用。但是，这种方式只是实现了人机界面的开放，数控系统的核心部分仍然是封闭的。

将 NC 板卡嵌入到通用 IPC 机中，既可以借助 IPC 机实现人机界面的开放，又可以借助 NC 板卡的可编程序能力实现系统的核心部分开放。相较于前一种方式，此种方式使得数控系统的建造更加开放和快捷，所以得到了研究人员，特别是国内研究人员的广泛采纳。但是这种数控系统的开放性高度依赖于特定的 NC 板卡（如基于 DSP、PMAC 运动控制卡等平台的板卡），并且只能实现系统核心的局部开放。

基于通用 IPC 机的软件化数控方式中，数控系统的基本功能完全由软件实现，系统的硬件部分由通用 IPC 机、I/O 通信设备及伺服驱动设备等标准化的硬件设备构成，从而使得从人机界面到系统核心、从软件到硬件的全方位开放提供了可能。并且，这种方式也为数控系统能够不断地吸收计算机软硬件最新成果创造了条件，也更有利于数控系统性能的提高及更新换代，因而被认为是开放式数控系统的重点发展方向和趋势。作为一项重要的战略技术，发达国家已纷纷投入了巨大的人力、物力对其展开深入的研究，如前文述及的 OMAC、OSACA、OSEC 研究计划就是典型代表。

1.5　研究项目来源

本书涉及的研究内容属于国家自然科学基金项目（51275485）的基础理论部分。该部分项目设计要求是利用 IPC 机和通用操作系统，构建一种全开放性的多轴联动控制的 Open-CNC 制造系统，为后续高速加工模块的添加和 CAM/CNC 一体化制造系统的实现作基础理论准备。

1.6 本书目的及其理论意义

传统的数控系统是不开放的。如 FANUC、SIEMENS、ABB 等公司生产的数控系统一般都是封闭的。这使得第三方生产商无法开发相关的应用，数控系统中无法应用最新的计算机软硬件技术。而对系统功能的许多扩展或改变都必须求助于系统供应商。

通用计算机的快速发展，为数控系统的开放奠定了技术基础，使数控技术从传统的封闭模式走出来，并融入主流计算机中，且随主流计算机的迅速进步而快速发展。

“数控系统的开放体系结构”可使 CNC 有效地应用于不同的计算机系统平台上，特别是运行于广泛应用的 Windows 操作系统平台上；还可以与其他系统互操作，提供与用户交互的统一界面。它具有性价比高、系统维护方便、更新换代升级快、配套能力强、共享计算机资源，以及便于用户二次开发等特点。

开放式数控系统及开放式数控机床的研究是当前数控技术“带有变革性”的重大发展方向，世界先进工业国家在此项目上都投入了大量的人力和财力。开放式数控技术的意义主要有以下几个方面：

1）数控系统具有开放性，即可重构、可扩展、可移植、可伸缩的特点，使 CNC 系统更加灵活。对 CNC 制造厂商而言，将大大缩短系统的开发周期，更易满足客户的不同需求。对最终用户而言，则可获得功能更强大、价格更便宜的 CNC 系统，并且升级和集成更为容易。而无论 CNC 系统制造厂商还是最终用户，都可减少对系统控制器卖方的依赖性，可集成自己独特的软件到不同系统中，随时扩展系统在构成 FMS、CIMS 等需求方面的能力。另外，系统的维护更为方便。

2）能适应数控编程加工标准的最新发展变化，可将现在广泛应用的 G、M 代码标准（ISO6893）逐渐过渡到采用面向对象的编程加工标准 STEP-NC（ISO14649）。

3）数控系统具有智能化。由于开放式数控系统具有可重构、智能化特点，因此可以发挥制造厂和用户的聪明才智和积极性，将它们的创新技术和工艺诀窍应用到开放式数控系统中，为数控设备的应用和发展提供了更广阔的空间。

4）CAM 和 CNC 之间的功能重新划分（CNC 中嵌入 CAM 功能），使 CNC 与其他 CAX 系统能进行双向通信，便于修改调整和更具有自适应性。

5）加工质量和效率大大提高。CNC 控制器具有规划功能，改变了被动执行者的地位。

6）使数据共享，促进网络制造技术的发展。

数控技术是现代制造技术的基础，数控技术是衡量一个国家经济实力和工业化水平的重要标志之一。我国是一个机床生产和应用的大国，但数控技术的应用水平还不高，严重制约着我国制造业水平的提高。国际上的相关开放计划对我国数控技术的发展提出了严峻的挑战，同时也带来了机遇。因为开放计划的实施，把世界上所有的系统开发商都推到了同一起跑线上。因此我们应该抓住机遇，坚持走数控系统开放化的技术路线，跟踪掌握数控技术的最新进展，并解决其中的关键技术，为新一代的国产开放式数控系统的研制打下坚实的基础。

1.7　主要内容安排

本书主要进行的是基于 IPC 机的软件化开放式数控系统关键技术的研究，涉及的关键技术指的是开放式数控系统的开放技术及多轴系统架构、NC 代码解释器、Soft-PLC 开发系统、Soft-PLC 运行系统、五轴样条实时插值等的原理及实现方法。本书具体研究了以下几个方面的内容。

（1）开放式多轴软数控系统平台及实现技术研究　分析支撑开放式数控系统的各种接口标准及软硬件平台，合理选择工业标准和成熟的商品化技术，搭建多轴系统的软硬件开发平台。硬件采用标准接口，软件参考 OMAC API 协议，采用 COM 组件，实时动态链接库技术构成模块化的软件库及采用有限状态机等技术，实现多轴软数控平台。

（2）开放式多轴软数控系统基本功能研究　实现数控系统的基本功能，包括数控系统人机界面、加工代码解释器及平面刀具补偿等功能的实现。

（3）Soft-PLC 技术研究　针对开放式数控系统，参考国际标准 IEC61131-3，研究并实现基于 IPC 机的 Soft-PLC 编程开发系统及 Soft-PLC 运行系统。

（4）五轴联动插值研究　插值功能是数控系统的核心，其种类很多，如空间直线、空间圆弧以及抛物线等。本书研究了现行五轴数控系统中常见也是必备的线性插值功能的实现及其加减速处理过程。除此以外，还将在 NURBS 曲线理论的基础上采用一种全新的适合于五轴联动的样条插值方法，并将对其在开放式数控系统中的实现方法进行了详细介绍。

（5）实验研究　利用以上技术构成一套数控系统并将其应用于三轴及五轴数控铣床系统，并进行了系统调试及切削实验，以验证其加工能力及开放性和可靠性。

第 2 章　Open-CNC 基础理论与系统架构

第 1 章曾提到，在基于通用 IPC 机的软件化数控方式中，数控系统的基本功能完全由软件实现，系统的硬件部分由 IPC 机、I/O 通信设备及伺服驱动设备等标准化的硬件设备构成，从而使得从人机界面到系统核心、从软件到硬件的全方位开放提供了可能。为达到此目的，就必须规定系统的接口标准，选定符合数控实时性要求的操作系统，并采用符合国际标准的通信及伺服驱动设备。

本章主要讨论实现软件化开放式数控系统的相关软硬件技术，并确定系统的总体架构及组成。

2.1　接口标准

开放式数控系统都采用模块化结构。数控系统内部模块与模块之间的相互通信，以及数控系统与外界（如与伺服系统）的信息传递都是通过相应的接口实现的。只有接口符合国际标准或行业事实标准，多个厂商的产品才能协调地组合在一起，从而满足可互换、可扩展、可移植和互操作等开放性要求。因此，真正的开放式数控系统，其接口必须符合标准。

数控系统的接口可分为两组——外部接口和内部接口[68]。外部接口包括数控系统与企业管理网的接口、与伺服系统及 I/O 设备的接口及数控（Numerical Control，NC）加工程序接口。目前各种外部接口都有相应的国际标准或行业事实标准：NC 程序接口标准有 ISO6983、STEP-NC；与伺服系统接口的标准有 +/−10V 模拟电压、SERCOS（Serial Real-time Communications Standard）、MACRO、FireWire 等；与 I/O 设备接口标准有 Profibus、Interbus、CAN 总线等；与企业管理网接口主要采用局域网。开放式数控系统是由模块化、可复用、可重构的软件构件构成的，所谓内部接口，就是系统的构件之间交换信息的接口，也叫做应用程序接口（API）。

数控系统与伺服系统的接口主要有两种：模拟接口和数字接口。

自数字式伺服系统问世之后，模拟接口的缺陷变得越来越明显，因而从 1986 年开始，欧美等发达的工业化国家便开始了数字式接口的研究，并逐渐形成了 SERCOS（IEC61491）、MACRO、FireWare（IEEE1394）等通信协议。

MACRO 是由美国 Delta Tau 公司开发，采用光纤或同轴电缆为传输介质构成

环状通路，传输速率最高为 125Mbit/s，支持位置、速度、转矩等多种伺服控制模式，每个环路可连接 256 个从站。虽然 MACRO 也有其他厂商支持，并非专有总线，但是它的实施协议并未成为国际标准，因此其适用范围和接受程度受到了一定的限制。另外 Firewire 也并非运动控制总线，而是一个物理层的定义，需要生产厂家增加协议、信息内容、数据格式、编码以构成基于 Firewire 的运动总线。它以双绞线为传输介质，采用树形结构，支持多个主站。每个主站可带 8 个从站，传输速率达 400Mbit/s 以上（各厂家稍有不同），支持位置、速度、转矩控制模式。由于基于 Firewire 的总线只有物理层符合国际标准 IEEE-1394，厂家添加的部分没有统一的标准而各不相同，因此各厂家的产品存在难以互换或互操作的情况。SERCOS 是用于运动控制与伺服系统之间的高速串行总线接口和数字交换协议，在 1995 年被确认为 IEC61491 国际标准。它不仅能够实现工业计算机与数字伺服系统之间的实时数据通信，而且它还对 I/O 功能作出了相应规定，可以同时完成 PC 机与 I/O 设备之间离散数字信号的实时通信。

最初，在 PC 机 + 运动控制卡的局部开放式数控系统中，有些具有 SERCOS 接口的运动控制卡实际上就集成了 SERCOS 主站功能[69]。另外，也有厂商提供独立完成 SERCOS 主站功能的产品。SERCOS 通信协议的处理是由主、从站上的 ASIC 芯片完成的，此时运动控制与伺服系统之间就是以该芯片上的双端口 RAM 作为接口进行数据交换的。

随着技术的进步，完全符合 SERCOS 标准的 SoftSERCANS 产品出现了。SoftSERCANS 实际上是通过向开发人员提供软件接口以方便使用 SERCOS 协议的一种技术，可以将其看做是安装在主站卡上的软件，作用就像 PC 机上的打印机驱动程序，可以在网上免费获得。它以 DLL（动态链接库）的形式为运动控制和伺服系统提供了 API 作为它们之间进行数据交换的接口。它的出现不仅使原来完成 SERCOS 的硬件得到简化，降低了 SERCOS 标准的实现成本，而且使原来较为复杂的接口初始化过程相对容易，开发者可以在数控系统中更加方便地采用 SERCOS 标准。该类通信卡的生产厂家有 Rexroth Indramat 、Sci-worx 、AUTOMATA 等。

本书采用 SoftSERCANS 作为 CNC 与伺服驱动器之间的接口，因为 SERCOS 已经成为国际标准，通过 SoftSERCANS 技术实现 SERCOS 标准更加简便；另一方面，它同时可以完成 PLC 和 I/O 系统之间的通信，使整个系统的结构变得更为简单。总之，工业伺服现场总线的迅速发展及标准化的实现，为数控系统的开放性奠定了坚实的基础，它也是软件数控技术发展的又一个重要推动力。

只有在数控系统的控制功能被细分成很多单元，且各功能单元之间的接口被详尽地定义的情况下，不同的软件供应商才能提供具有各自特色的功能单元模块，数控系统才能真正做到不依赖于特定的供应商。所以，模块化是实现开

放式数控系统的前提。

为了将数控系统内部的各模块协调地组合在一起，构成开放式数控系统，需要一组完整且标准的 API 集合。这就要求有一个国际标准规定数控系统内部所有模块（包括实时部分，如运动控制、轴控制、PLC 功能等）的接口。但目前还没有这方面的标准。

始于 1992 年欧共体的 OSACA 计划是开放式数控系统领域的第一个国际化研究工程，随后，日本的 OSEC 工程和美国的 OMAC 工程也做了相似的努力。它们都采用了模块化的结构，并规定了系统集成的内部接口——模块 API。这三个国际化的工程在开放式数控系统领域最具影响力，而它们也都在为尽快形成开放结构控制器的国际标准而努力。

这三个国际化的工程在实现开放式数控系统方面都有很多有效的方法，但我们无法将这三个计划的结果简单地集成到一个统一的开放式数控系统中。而我们又非常需要一个通用的、能被各方所接受的且被大多数数控系统一线厂商所支持的 API 接口国际标准，否则就会形成这样的情况：用很多“不依赖于特定供应商”的不兼容的数控系统替代了很多“厂商专有”的不兼容的数控系统，最终仍然达不到彻底开放的目的。

详细的比较表明：在技术上，OMAC 更优秀一些；另外，OMAC 是由美国三大汽车公司联合发起的，且得到 FANUC、SIEMENS、Allen-Bradley 等著名数控系统生产厂商的支持。因此，无论是从技术还是从市场的角度看，开放式数控系统的国际标准都最有可能在 OMAC 的基础上形成。

开放式数控系统应该是模块化的，且其外部接口及内部接口必须遵循一系列的国际标准或行业事实标准，没有全面的国际标准就不可能有真正的开放式数控系统。

鉴于此，本书研究的开放式数控系统采用模块化结构，数控系统内部模块与模块之间的相互通信，以及数控系统与外界（如与伺服系统）的信息传递都是通过相应的接口实现的。只有接口符合国际标准或行业事实标准，多个厂商的产品才能协调地组合在一起，从而满足可互换、可扩展、可移植和互操作等开放性要求。因此，真正的开放式数控系统，其接口必须符合标准。

2.2 系统实时性实现原理

数控系统是一种典型的实时多任务计算机控制系统。其中的很多任务，如加减速运算、插值运算及位置控制等，都是实时性很强的任务。目前，很多高速、高精度数控系统的插值周期和采样周期已经达到了 1ms 以下。如不能在规定周期内完成插值计算或位置控制任务，加工过程就会出现断续和停顿，从而

影响工件加工质量，并减少刀具使用寿命。另外，如用户通过控制台发出的急停指令，必须在给定的最短时间内作出响应，否则就会危及设备和人身安全。因此我们说，数控系统是一种硬实时系统。

既然数控系统是一种硬实时系统，那么它所采用的操作系统当然必须具备实时性。

2.2.1 何谓实时

实时系统的特点在于：一个正确的运行不仅取决于结果的准确，更取决于实现的时间。需要注意的是：“实时”并不意味着“快”，它指的是系统的时间响应特性。换句话说，实时性的衡量标准不是系统的平均响应时间，而是最坏情况下的响应时间。实时系统有时被进一步划分为硬实时系统和软实时系统。硬实时系统对响应时间的要求是严格的、绝对的；而软实时系统允许有一些小的误差。某些观点认为“软实时”的说法是自相矛盾的，在以下介绍中凡涉及“实时”都指的是硬实时系统。

实时有以下三个方面的含义：

1）实时意味着绝对可靠。

2）实时系统必须满足时间限制，以避免失败。

3）实时系统的确定性是指在固定的时间里完成规定任务的能力。

除了确定性，实时系统通常还有一些其他要求：

1）一个具有很多优先级的多线程优先级调度器。

2）可预测的线程同步机制。

3）具有优先级继承。

4）快速的时钟和定时器。

2.2.2 实时与非实时系统的比较

通用操作系统（如 Windows 操作系统）就是非实时操作系统，其特点为：①高优先级任务可以被低优先级任务抢占；②具有额外的后台程序、屏幕保护程序、防毒程序和磁盘管理程序等；③外围设备中断。

而实时操作系统的特点为：①调度器确保高优先级的程序首先执行，并不被低优先级程序中断；②对所有的任务直接控制；③外围设备中断不能打断高优先级程序执行，除非其优先级更高。

2.2.3 实时系统间的比较

常见的实时操作系统有以下几种。

（1）DOS 操作系统 DOS 操作系统是典型的单进程、单线程的单任务实时

系统；只能进行单任务编程，人机交互和网络功能差，且缺乏技术支持。

（2）嵌入式操作系统　嵌入式实时系统，如 Wind River System 的 VxWorks、QNX Software Systems 的 QNX、CMX System 的 CMX、Microsoft 公司的 Windows CE 等，在工业控制领域应用较广泛，且实时性能能满足硬实时控制系统的要求[70-72]。这类实时操作系统的应用总是结合上位机构成双 CPU 系统协调工作。上位机操作系统中主要完成一些非实时或弱实时任务，而实时操作系统则专注于实现实时要求高的任务。因此限制了这类实时操作系统的应用，即多用于一些嵌入式的控制系统中。同时这类系统提供的开发环境单一、支持工具较少，以及不具备很好的兼容性等特点，也就影响了其在开放式控制系统开发中的应用。

（3）基于 Linux 的实时操作系统　Linux 本身并不具有较好的硬实时性能，但它是一种源代码开放的操作系统。因此开发人员在获得 Linux 操作系统源代码的基础上，对其代码进行适当的修改，形成满足一定硬实时性的操作系统软件。目前基于 Linux 的实时操作系统有 FSMLabs 公司的 RTLinux，LynuxWorks 公司的 LynxOS 和 Blue Cat Linux，以及 MontaVistaSoftware 的 Hard Hat Linux。RTLinux 是一个能运行在多种硬件平台上的 32 位硬实时系统，且具有优良的实时性能和稳定性。目前，国内一些研究机构也采用这种软件平台开发自己的控制器[73]。但是这类操作系统安全性能不能完全得到保证，也不能支持多数的硬件系统，且目前基于 Linux 的软件资源还不够丰富。

（4）Windows 实时扩展操作系统　Windows 凭借其 Win32API 的广泛应用及开放的体系结构成为工业控制中操作系统较为理想的候选者。但是这个系列的操作系统（Windows NT/2000/XP）实时性能较差，其中断响应时间延时高达 1 ~2s，根本就不能满足数控系统的实时响应要求。因此许多软件开发商开发了针对 Windows 的实时扩展，诸如美国 VenturCom 公司的 RTX、TenAsys 公司的 Intime、Nematron 公司的 HyperKernel。它们都在保全 Windows 原有功能的基础上，提供优异的实时性能。并且这类产品经过了 OMAC 工作组严格的实时测试，结果表明，它们的实时性能完全满足了数控系统的要求[74]。

RTX[75]是内置于 Windows 的实时操作系统，可以认为是 Windows 的一个子系统 RTSS（Real-Time Subsystem），如图 2-1 所示，它不影响 Windows 的原有功能，却增强了其实时性能。实际上，VenturCom 公司获得了 Windows 的源代码，对其硬件抽象层 HAL 进行了实时扩展。该实时硬件抽象扩展层隔离了 RTSS 和 Windows 之间的中断，RTSS 进程运行时 Windows 的中断被屏蔽，但 Windows 不能屏蔽 RTSS 管理的中断。在单 CPU 的情况下，所有 RTSS 线程的优先级高于所有 Windows 线程优先级（包括 Windows 管

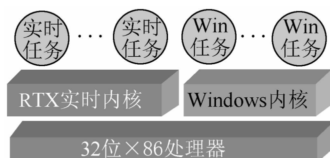


图 2-1　Windows + RTX 示意图

理的中断和延迟过程调用 DPCs）。RTX 还可以保证任意线程的最差响应时间为 50μs。

2.2.4　实时系统的确定

通过上面的分析，我们倾向于采用基于 Windows 的实时扩展系统作为开发数控的软件平台。这样可以最大限度地利用 Windows 操作系统的开放性特征，而且 Windows 经实时扩展后，数控系统所要求的实时性能也获得了充分的保障。因此，本系统采用 Windows + RTX 作为开放式数控系统的操作系统。

2.2.5　RTX 实时系统的利用

RTX（Real-Time Extension）是美国 VenturCom 公司开发对 Windows 系统的实时扩展子系统。由硬件抽象层 HAL（HardWare Abstract Layer）扩展、实时子系统 RTSS（Real-Time Subsystem）及实时开发工具库组成，如图 2-2 所示。

RTX 实现了确定性的实时线程调度、实时环境和与原始 Windows 环境之间的进程间通信机制以及其他只在特定的实时操作系统中才有的对 Windows 系统的扩展特性。

RTX 被实现为一套库的集合（动态库与静态库），RTSS 作为 Windows XP 的内核设备驱动与 HAL 扩展（见图 2-2）。子系统实现前面提到的实时对象和调度器。通过一套被称作 RtWinAPI 的实时 API（RtWinAPI 同时也被 Windows CE 和 Phar Lap ETS 支持），这套库提供了对这些对象的访问方法。RtWinAPI 可以被标准 Win32 环境和 RTSS 环境调用。虽然在 Win32 环境中使用 RtWinAPI 不能提供在 RTSS 下的确定性，但是却可以允许应用程序在更加友好的 Win32 编程环境中开发而不是 DDK 环境。将 Win32 程序转化为 RTX 程序只需要重新链接一套不同的库而已。Windows XP 服务控制管理器直接将 RTX 进程和动态链接库（DLL）的可执行映像装入内核的不分页内存中。

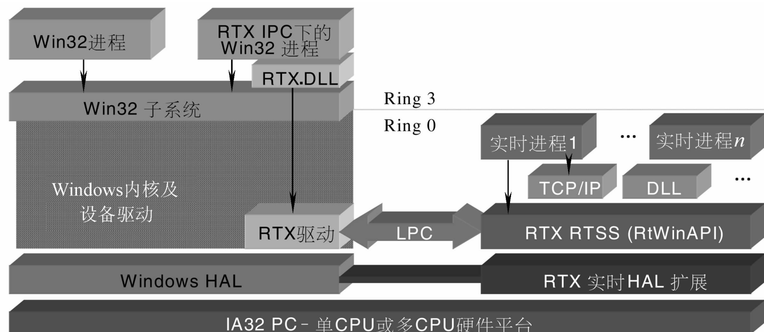


图 2-2　RTX 系统架构

（1）实时硬件抽象层（HAL）　HAL 是 Windows XP 系统提供的可被用来进行修改和扩展的资源的一部分。RTX 修改 HAL 有以下 3 个目的：①在 Windows XP 和 RTX 线程之间增加独立的中断间隔；②实现高速时钟和定时器；③实现关闭处理程序。中断隔离意味着 Windows XP 线程和 Windows XP 管理的设备不可能中断 RTSS，同时 Windows XP 线程也不能屏蔽 RTSS 管理的设备。HAL 通过控制处理器级的中断屏蔽满足这些条件。当运行 RTSS 线程时，所有 Windows XP 控制的中断都被屏蔽掉。当 Windows XP 线程请求设置中断屏蔽时，作为实际管理中断屏蔽的软件，HAL 确保没有任何 RTSS 中断被屏蔽。Windows XP 提供的计时器的最小计时单位为 1ms。RT-HAL 将其降到了 100ns 并且提供了同步（与计时器）的时钟。

（2）RTSS 调度器　RTSS 调度器采用抢占式策略实现优先级，它可以提升优先级以防止优先级倒置。RTSS 环境提供了 128 种优先级，序号从 0～127，0 代表最低优先权。RTSS 调度器总是在准备运行的线程中运行优先级最高的（当多个准备运行的线程处于同一优先级时，等待时间最长的线程最先运行）。RTSS 线程会一直运行直到一个高优先级的就绪线程抢占它，或者它自动释放处理器进入等待状态，还有一种情况是分配给它的时间片用光（缺省值是无限）而另一个同优先级线程已经就绪。

调度器在编码设计阶段就被设计为满足实时处理的需要。最重要的是它的操作是低延迟的，并且不受它所管理的线程数影响。每个优先级都有自己的等待队列，是一个双向链表。这就使得对于链表的插入（表尾）和删除（表的任何位置）操作的执行时间独立于链表中的线程数。一个数组会纪录哪些链表当前是空的，这个数组是由高速的由汇编代码写成的子程序进行维护的。

当一个 RTSS 线程运行时，所有 Windows XP 管理的中断以及任何拥有低优先级的线程所管理的中断都一律被屏蔽掉。相反地，所有拥有高优先级的线程所管理的中断都不会被屏蔽，并且允许其打断当前线程。除了这些设备中断，其他可以导致当前线程被中断的机制包括一个使得拥有高优先级的线程就绪定时器的到期，一个标明高优先级线程正在等待的同步对象信号（正在运行线程的同步对象）。

为了解决线程抢占的问题，RTSS 采用了经典的优先级提升的解决方案。当一个低优先级线程拥有一个高优先级线程等待的对象时，在它拥有对象的时间内会被自动提升到较高的优先级别。

（3）动态链接库　提到 Win32 就不得不提到 DLL 库。RTSS 支持 Win32 DLL API（LoadLibrary 函数，GetProcAddress 函数）。现在，所有在 RTSS 的 DLL 中的静态和全局变量都被链接到其上任何 RTSS 进程所共享。

（4）使用 Visual Studio 创建 RTX 应用程序　通过 RTX 提供的头文件和库

函数，RTX 应用程序可以像其他任何 Windows 应用程序一样被 Microsoft Visual Studio 编译和链接。Application wizard 可以用来更改工程设置和创建源代码框架。从 RTX5.1 开始，Visual Studio debugger 可以像调试其他 Win32 进程一样调试运行 RTX 进程。进程与线程的变量和断点可以被设置和管理。

2.3　OMAC 工程及 OMAC API

OMAC 用户组是一个致力于提升控制器技术水平的工业研讨会[76]。OMAC 的目标之一是定义一个最终被提交为标准的 API 协议。

OMAC API 采纳基于构件的软件思想实现“即插即用”的模块化，并使用接口类规定 API。为适应分布式通信，基于构件的技术采用代理存根模式来处理进程间的方法调用。OMAC API 包含不同粒度和类型的可复用的即插即用插件——构件、模块和任务——每一种都拥有一个唯一的有限状态机（FSM）模型，以便以一种可知的方式实现构件的协作。术语构件（component）是指在可复用的软件段，它是构建应用程序的最基本的“建筑块”，而术语模块（module）是指一个包含一些构件的包容体。任务（task）是用来封装可编程的功能行为的构件，这里的可编程的功能行为是指一组从头到尾顺序执行的操作步骤，如启动、停止、重启、暂停和恢复等。任务包括暂态任务（如 RS274 程序）以及处理特定的控制器需求的驻留任务（如机床回原点、换刀、急停等）。

为了集成构件，需要有一个框架（framework），而构件框架是构件实例“即插即用”的支撑结构。OMAC API 使用微软的 COM 作为目前的构件框架，在这个框架内开发构件的数控厂商可以专注于专用控制程序的改进。使用 COM 构件框架的最大问题是必须使用 Windows 操作系统，而 Windows 操作系统不具备硬实时性，但这个问题可以通过使用第三方 Windows 实时扩展软件来解决。

构件框架（framework）是构件实例“即插即用”的支撑结构。通过一定的环境条件和交互规则，构件框架允许一组构件形成一个“孤岛”，独立地与外部构件或其他框架交互和协作，因此构件框架及其内含的构件也可以视为一个构件，于是构件通过不断的迭代和合成，构成一个结构复杂的应用系统。目前，有多个组织和公司制定了构件基础设施的标准或开发了相关产品，也为构件、构件框架和接口建立了模型和技术规范，其中 OMG CORBA、Microsoft COM/DCOM（或.NET）以及 Sun JavaBean 占主导地位。

2.3.1　动态链接库（DLL）技术

采用面向对象编程可实现源代码一级的复用，在软件重用领域中属于白盒复用，需要根据用户需求对已有程序代码进行适应性修改后才能使用。与白盒

复用相对的软件复用方式是黑盒复用，它主要是基于二进制代码的复用，包括可执行程序的复用和基于函数库的复用。黑盒复用是指对已有程序或构件不需作任何修改，直接复用，这也是理想的复用方式[16,77]。

函数库简单地说就是一个文件包含了一些编译好的代码和数据，这些编译好的代码和数据可以供其他的程序重复使用。例如：Microsoft Visual C ++ 提供的 MFC 类库及 Borland C + + 提供的 OWL（Object Windows Library，面向对象的窗口库）类库，它们为 Windows 应用程序的开发提供了最基本的代码，开发人员直接使用这些类库就可以做出一些基本的 Windows 应用程序来。程序函数库可分为两种类型：静态函数库和动态函数库。静态函数库是在程序执行前就加入到目标程序中，而动态加载函数库则可以在程序运行的任何时候动态加载。动态库特别适合在系统需要模块的功能时才进行动态加载的场合，并符合数控系统的开放性需求。

以模块作为开放式数控系统的基本单元，使数控系统能够适用于不同的应用需求（如车、铣、钻等），且模块必须具有可复用的特征。采用函数库的形式开发数控功能模块是一种理想的选择。针对数控系统构成模块所完成任务的实时性需求不同，可以将模块分为非实时和实时模块两类，并分别采用 COM 组件和实时动态链接库 RTDLL（Real-Time Dynamical Linked Library）的形式实现。

（1）COM 组件　COM 是 Microsoft 提出的组件标准，它不仅定义了组件程序之间进行交互的标准，而且提供了组件程序运行所需要的环境。COM 组件是符合组件对象模型标准的、运行在 Windows 环境下的一种特殊动态链接库。COM 标准要求组件具有统一的规格和规范的使用方法，并将特定的功能和行为封装在内部，再通过规范的接口对外提供服务。COM 组件的主要特征有：面向对象的特征、客户/服务器特性、语言无关性、对进程的透明性和它的可重用机制[78]。COM 组件自身具有的优势，为实现开放式数控系统的可互换性、互操作性、可伸缩性等要求提供了充分的技术保障。

COM 组件可以利用 Visual C ++ 开发，它提供了许多简化 COM 组件开发的编程工具。例如：利用 COM 开发向导能够自动生成骨架代码和基础结构，所以开发人员无需过多地关心像全局唯一标识符 GUID（Globally Unique Identifier）、接口派生、类工厂等一些细节问题，而把注意力放在定义接口、编写接口的实现代码上，再通过接口向外提供服务[79,80]。

数控系统的非实时任务包括数控程序的输入与编辑，系统信息与加工信息的显示、译码、刀具补偿等。这些任务并不要求很强的实时性，使用 Windows 的定时器就可以满足要求，因此利用 COM 技术可将这些任务编制成为在 Windows 环境下运行的 COM 组件。

（2）实时动态链接库 RTDLL　数控系统中实时任务是指加减速处理、插值

运算、位置控制、离散逻辑控制等。这些强实时性任务必须在 RTX 环境下完成，才能满足其实时性要求。RTX 支持面向对象编程技术，并在 Visual C + + 编程环境提供了一个向导工具，具有 Windows 开发经验的人员可快速掌握 RTAPI 并进行实时编程。对于以上强实时性任务，我们采用 RTX 提供的动态链接库 RTDLL 技术来完成。RTDLL 类似于 Windows 的动态链接库，可以在 RTX 环境下动态加载，而 RTDLL 所具有的实时性是以 RTX 的实时 API 为保障基础[81,82]。为实现实时编程，RTX 提供了三类实时 API（real-time API）：第一类为 RTAPI，是由 RTX 单独提供的一套实时 API，这类函数均以字母“Rt”为前缀；第二类为 RTX 支持的 Win32 API，RTX 对部分 Win32 API 函数的内部实现进行了修改，使之具有实时性能，且该类函数在 Windows 和 RTSS 环境下的应用形式完全保持一致；第三类为 RTX 支持的 C 运行时库函数，由 RTX 提供的具有实时性能的 C 运行时库函数，用来替代相对应的 Windows C 运行时库函数。

RTDLL 可以采用隐式连接或显式连接方式处理。隐式连接是指在应用程序启动时立即加载 RTDLL；显式连接方式则可以在应用程序运行时，根据需要决定装载哪个 RTDLL，以及在什么时候加载或卸载。显式加载的函数为 LoadLibrary（LPCTSTR lpLibFileName），通过 lpLibFileName 指定所加载实时动态链接库的名字。

2.3.2　客户机/服务器通信模式

OMAC API 采用客户机/服务器的模式来实现对象间的通信。在这种模式下，一个对象作为服务器，而这个对象的使用者叫做客户机。一个对象既可以作为客户机，又可以作为服务器来使用。对象间的协作是通过由客户机向服务器发出请求，服务器响应客户机的请求来实现的。对于 OMAC API，客户机调用类方法以实现上面描述的协作行为。客户机使用存取方法操作数据，存取方法通过对数据的抽象隐藏了数据的具体物理意义。

标准的客户机/服务器请求引发一个同步操作的执行。同步执行在客户机/服务器中做一个往返：客户机发布一个请求后服务器接到一个方法调用，执行相应的方法实现，并向客户机送回一个应答信号。OMAC API 定义了三种类型的客户机/服务器请求：参数请求、指令请求和监视器请求。具体如下：

1）参数请求是利用方法调用对对象的数据成员进行读写操作，这种操作可在一个往返中完成。参数请求不需要状态空间逻辑。

2）命令请求传送的是事件，它能引发服务器的状态转移并产生一个新的服务器状态。命令请求可能运行一个或多个循环，如轴模块完成机床回原点操作。响应这类请求时，客户机和服务器之间的协调需要状态空间逻辑，且基于服务器的有限状态机模型。

3）监视器请求协调多个模块的执行，如轴模块的机床回原点操作。监视器请求需要状态空间逻辑。方法向轴模块发送一个事件，而如何执行取决于它的状态空间逻辑。每一个伺服周期调用一次方法以实现轴模块的周期执行：在每个周期，它取得数据（位置指令值和实际反馈值）并产生新的输出点。

2.4 有限状态机在 CNC 中的实现机制

有限状态机（FSM）由有限的状态和相互之间的转移构成，在任何时候只能处于给定数目的状态中的一个。当接收到一个输入事件时，状态机产生一个输出，同时也可能伴随着状态的转移[83]。有限状态机由以下要素构成：

（1）状态 状态是一个有限状态机的生命周期中满足某种条件，执行某些动作（Action），或等待某些输入事件发生的一个阶段。它能持续一段时间。

一个状态对应地执行某些活动（Activity）。活动就对应一个嵌套的状态机。

（2）转移 转移是从一个状态结点到另一个状态结点（目标状态）的移动。

（3）事件 事件是一个在时空中显式出现的特定现象，它可以触发状态转移。事件可以分为好几类，如信号事件、定时事件等。

（4）动作 动作是对象在状态发生转移时所作的一系列处理操作。动作（Action）是原子的，不可被中断的。它的执行时间是忽略不计的，它在状态进入和推出的时候执行，或在转移时执行。

在有限状态机中，还有一种特殊的转移，即自身状态转移，它是源状态等于目标状态的转移，如一段运动程序（直线、圆弧、NURBS 样条等）在轴组中进行插值。

FSM 经常被应用于反应式系统的建模，而反应式系统对外部和内部事件作出反应，以事件驱动方式工作。在确定性的反应式系统中，输入事件的顺序和数值决定了系统响应的顺序和数值。

数控系统根据输入的信息（如数控程序、操作面板的输入、传感器反馈信息）控制机床移动，实现加工操作（如轴运动、换刀、停止机床等），其行为是可预见的，属于典型的反应式系统。因此，我们将外部输入的信息表示为 FSM 的输入事件，将机床的加工操作表示为 FSM 的动作，采用 FSM 作为机床的行为模型[33,84]。

在实际应用中，机床的行为非常复杂，其状态数目庞大，需要将整个系统按照层级方式分解得到一系列子功能模块。整个系统的行为被分解为子系统的行为，并用单独的 FSM 描述，然后再把这些模块集成起来构成系统，就得到了由层级式 FSM 所表述的整个系统的行为模型。本系统中的 FSM 如图 2-3

所示。

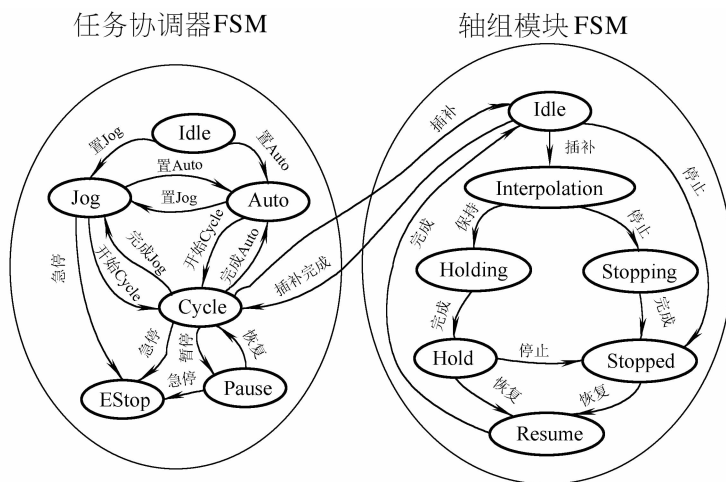


图 2-3　系统中的 FSM

FSM 不是独立存在的，它依附于构成数控系统的各个构件。在分层 FSM 中，不同层级的 FSM 之间通过构件接口进行通信。为了调用一个低层 FSM，包含上层 FSM 的构件发送一个事件到包含下层 FSM 的构件接口；当下层 FSM 到达终止状态时，发送一个终止通知事件到上层 FSM 的接口。

当系统部件、功能或过程顺序有变化的时候，加工系统的可重构是必需的。每种情形都需要控制流的改变。使用 FSM 机制，控制流能与机械分离开来而被单独改变。另外，FSM 为系统开发者提供了一种指定专用控制流的方法。采用这种方法，系统重构所需的努力和技巧都可以显著减少。

加工系统重构包括加工部件的重组和过程顺序的改变。在一个开放式数控系统中，一个机械系统被模块化为不同的部件，它们中的每一个都完成一些规定的功能。机械重组意味着使用新的或不同的部件，或将部件以不同的方式组合在一起。例如：将一个钻削系统变为车削系统，或者将一个三轴轴组分成一个两轴轴组和一个单轴轴组。过程改变是指使用带相同的部件组的同样的机床，而只是操作顺序发生了改变。也有很多时候组成部件和过程顺序都发生了变化。在实际应用中，最常见的变化是处理过程和（或）部件的一个子集发生变化。很少有整个系统或整台机床被重新设计或重新建造。因为在开放式数控系统中，不管是功能还是过程顺序都被模型化为控制流，所以通过改变系统的控制流可以很容易地重构加工系统。

在一个三轴铣削系统中，FSM 能被用来控制一个两轴轴组，一个单轴轴组和每一个轴。创造一个三轴轴组将引入一个新的控制三轴轴组的有限状态机，但每个轴的 FSM 还将保持不变。在数控系统中扩充 NURBS 曲线加工能力，相当

于增加系统的功能。但由于采用了 FSM 模型，因此只有与插值相关的模块如任务生成器和轴组模块需要修改，而其他的模块则不需要变动，这极大地减少了系统功能扩充的编程工作量。

2.5 SERCOS 接口原理与实现方法

2.5.1 SERCOS 接口简介

SERCOS（Serial Real-time Communication System）是用于数字控制器与数字驱动器之间的高速串行总线接口和数字交换协议。由德国主要 CNC 制造商、伺服系统制造商和研究机构共同发起成立的 SERCOS 协会提出，目的是定制 CNC 系统与数字伺服系统之间统一的数据交换接口，提供各厂商产品间的互换可能性。它于 1995 年被确认为 IEC61491 国际标准。同时 SERCOS 也对 I/O 功能作出了相应的规定，可以实现 PC 机与 I/O 设备之间离散数字信号的实时通信[85-88]。

SERCOS 接口通信系统是一个主站（Master）和若干个从站（Slave，数字伺服驱动器或 I/O 模块），通过光纤串行连接构成的环形网络。站间最大距离为 40m，可连接 254 个从站，传输速率可达 4Mbit/s、8Mbit/s、16Mbit/s，伺服周期可以设定为 62.5μs、125μs、250μs、500μs 的整数倍。在一个周期内发送和反馈的数据类型和数量是可以预设的，这种灵活性使得设计者能够通过改变周期、数据内容及驱动器数量来满足特定的应用需求。SERCOS 接口支持位置、力矩及速度三种控制模式。SERCOS 接口通过传送特定控制模式下设定的 S、P 参数实现数字控制器与数字驱动器之间的通信。S 参数是 SERCOS 接口标准参数，P 参数是各厂家产品参数（S、P 参数均存储在驱动器等从站内，故也可称为驱动器参数），对这两种参数的操作可实现具有读写系统的控制特性。

SERCOS 协议通信数据报文采用高级数据链路控制规程（HDLC）国际标准，其格式见表 2-1。数据报文边界标志 01111110 之间的数据为有效数据，数据校验段为 16 位，采用 CRC 循环冗余码，地址段和数据段定义取决于数据报文类型。SERCOS 协议规定了在光纤环上传送的三种数据报文。

第 1 种为主站同步报文 MST（Master Synchronization Telegram）：由主站在每个工作周期的开始时广播，作为每个传输周期光纤环中的时序同步信号。

第 2 种为驱动器报文 AT（Axis Telegram）：由各个从站（驱动器）分别将多种伺服信息实时反馈给主站，如伺服轴实际位置、实际转速、实际转矩、报警信号、诊断信号、状态应答信号、伺服参数以及 I/O 模块的采集输入等。

第 3 种为主站数据报文 MDT（Master Data Telegram）：由主站向从站发出的控制指令，如位置指令、速度指令、转矩指令、控制方式以及 PLC 的输出等，

在一个周期内规定的时刻发送一次。

表 2-1　SERCOS 通信协议

管理段		用户数据	管理段	
报文边界符	地址段	数据段（可配置长度）	帧校验段	报文边界符
01111110	8 位	8 位 $*n$（$n=1, 2\cdots$）	16 位	01111110

2.5.2　SERCANS 软件功能

SERCOS 接口的内部机理决定了在运行系统时，首先必须进行复杂的初始化操作。初始化时必须访问环上各站点的许多数据，然后进行时间片计算。为了使运动控制开发者更好地接受 SERCOS 接口，1996 年诞生 SERCANS 概念，其目的是将复杂的初始化过程封装起来。SERCANS 是一个主动式的 SERCOS 主站卡。在 SERCANS 卡上有个微处理器，由装入的软件执行 SERCOS 环的初始化和管理。在 SERCANS 卡上有个双口随机存储器，是 SERCOS 接口环和控制器间数据交换的接口。中断信号触发的方式同步控制器到指定的存储器入口地址读写数据。

SoftSERCANS 以软件形式实现了 SERCANS 的部分工作，并将 SERCANS 主控功能移植到了软件抽象层。SoftSERCANS 只需要使用一张被动式 SERCOS 主站卡，而主站卡仅仅是一张简单的具有一定机械、电气功能特性的数据传输卡。SoftSERCANS 就是驱动被动式主站卡的运行在实时环境下的进程，并利用一个动态链接库（DLL）向运动控制器提供通信接口。SoftSERCANS 运行在实时扩展美国 VenturCom 公司开发 RTX 子系统的 Windows 操作系统平台上，并采取某些措施解决了实时 Windows 平台的局限性（延时问题）和运动控制器高实时性要求之间的矛盾，使 SERCOS 接口具有几个微秒的内部实时能力[89]。

运动控制器开发者在掌握了 SoftSERCANS 提供的 DLL 函数及相关的参数设置后，就可以很方便地实现控制器和数字驱动器的实时数据交换，无需花费较多的精力在 SERCOS 接口的具体实施上。运动控制器运行后，首先要通过 SoftSERCANS 提供的 DLL 接口函数初始化 SERCOS 接口，然后进入 5 个通信阶段。

1）Communication Phase 0：闭合光缆环，光缆环闭合后进入下阶段。

2）Communication Phase 1：检测环中存在的驱动器。光缆环中每个驱动器或 I/O 模块必须通过地址认可。在确定所有环中的从站分别对应的参数地址后，进入阶段 2。

3）Communication Phase 2：参数化阶段，设定从站的控制模式（位置控制、速度控制、转矩控制等）和传送数据的格式和比例因子；MDT 报文、AT 报文内容的配置；计算各从站 AT 报文和主站 MDT 报文发送的时间段，并将这些通信

参数发送到从站。

4）Communication Phase 3：向各个从站传送执行命令。

5）Communication Phase 4：数据交换阶段。

在每个通信周期的开始，SERCOS 接口向从站发出 MST 报文作为每个周期的同步信号。从站接到 MST 报文后在规定的时间槽内以 AT 报文形式发送自己的实际值（反馈值）给主站。

SoftSERCANS 在 SERCOS 参数的基础上增加了一些系统参数（A、Y 参数），并存储在 SoftSERCANS 中。其中，Y 参数用于光纤环的配置，而 A 参数用于规定 SERCOS 接口驱动器的特性，每个从站都有一组自己的 A 参数。具体的设定方法和工具软件详见第 6 章。

2.6 Open-CNC 制造系统的架构

2.6.1 系统架构

基于前面的分析与讨论，可建立如图 2-4 所示基于 IPC 机的 software 型多轴 Open-CNC 数字化制造系统总体架构。

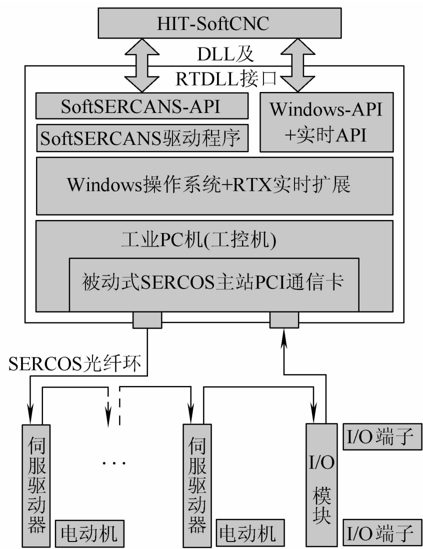


图 2-4 系统架构

2.6.2 系统组成

对于模块化的全软件型开放式数控系统而言，模块是构成系统的可重用基

本单元。但是，目前对于模块的划分还没有统一的标准，在确定模块的复杂度时，需要对开放程度、运行效率、系统复杂度和集成费用等影响因素进行综合考虑。模块粒度较大，则可重用性和灵活性较差；反之，粒度越小，开放程度提高，可提供的选择也越多。但是构成系统的单元数目过多，可能导致模块间的交互过于复杂，运行效率会降低，性能也变差。所以，模块划分时，除了以一定的独立功能为界限，还要遵循粒度适中的原则。

本书采用面向对象的方法，依据模块化的分解思想和层级式组织方式建立开放式控制器。分析通用数控系统功能[90]，总结现有系统控制结构的共同特征，并加以分类。将整个软件系统划分为以下几个基本功能模块，如图 2-5 所示。开放式控制器被分解为一系列功能模块，这些模块是分层组织的，上层模块向下层模块发出服务请求，并监控下层模块的运行；模块之间通过标准接口进行通信。

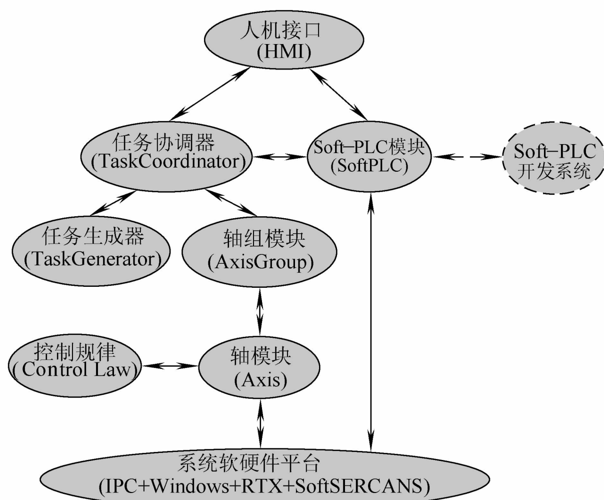


图 2-5　开放式控制器的模块化结构

(1) 人机接口（HMI）　负责操作者与控制器之间的交互，主要完成在系统运行前和运行中系统参数的修改和设定，如设定系统工作模式（自动、手动、点动，编辑 MDI 等）、系统初始化设定、坐标偏置设定，以及数控加工程序的编辑、加工和诊断信息的显示等。

(2) 任务协调模块（TaskCoordinator）　进行任务分配，负责系统内各模块的协调与调度。

(3) 任务生成模块（TaskGenerator）　也即任务生成器，根据一定的语法规

则对数控加工程序进行语法检查，完成译码工作。对加工程序中给出的各种信息分离提取，生成包含运动信息的运动段指令和逻辑控制指令，还包括对运动指令进行刀具补偿。

（4）轴组模块（AxisGroup） 完成加减速处理、插值任务，对译码得到的运动段按照一定的进给速度要求进行细分，进而得到单个插值周期内的直线段进给量；然后分解为相应轴上的位置目标指令，并输出给各个轴模块。

（5）轴模块（Axis） 接收来自轴组模块的指令，同时读取外部的反馈信息；再根据用户需求选择性地调用伺服控制规律完成位置控制或速度控制等功能，并将控制信息发送给外部执行单元。

（6）控制规律模块（Control Law） 负责伺服控制规律的计算，提供多种控制策略，如模糊控制、自适应控制、神经网络等。

（7）Soft-PLC 模块（SoftPLC） 又称为 Soft-PLC 运行系统，与任务协调模块进行信息交互，负责对外部输入和内部状态变量进行布尔运算得到相应输出和内部状态变量，并对外部输入输出设备进行控制，包括机电设备的起停、刀具的交换等。其中，通过任务协调器与 Soft-PLC 系统进行数据信息量交互的人机界面，包含面板控制量输入、各种状态显示及实时系统的控制。开关量输入如手动、自动、起动、急停、主轴正转、主轴停、主轴反转、进给循环、进给暂停等，其状态是作为 PLC 逻辑控制程序的输入量。显示灯如手动状态、自动状态、回原点、X 原点、Y 原点、Z 原点等，是 PLC 控制程序的输出量。

（8）Soft-PLC 开发系统 是带有调试和编译功能的 PLC 编程器。

图 2-5 中，Soft-PLC 开发系统是整个 Soft-PLC 系统的一部分，整个 Soft-PLC 系统由开发系统和运行系统组成，两者是既相互独立又联系密切的两个应用程序，并可分别单独运行。系统的运行过程是先用 PLC 开发系统编辑开发 PLC 控制程序，生成目标代码，然后由 PLC 运行系统运行这个目标代码，来实现对系统的控制。具体内容将在第 4 章论述。

2.7 小结

本章论述了基于 IPC 机的开放式多轴数控系统的接口标准及实现开放式数控的相关技术，即参考 OMAC API 协议、采用有限状态机模型及应用 SERCOS 等技术。最后确立了系统的软硬件平台及系统的总体架构与组成。

采用 Windows + RTX 作为数控系统的软件平台，既保留了 Windows 操作系统的开放性，又能满足数控系统的实时性需求。系统功能采用面向对象的 C ++ 语言，在 Windows 操作系统和 Visual C ++ 的开发环境下，采用模块化结构构建而成。

第 3 章　Open-CNC 制造系统的代码解译机理

3.1　概述

加工代码的解释器（对应 2.6 节的任务生成模块）是开放式软件化数控系统的关键技术之一，本书中 NC 代码解释器由任务生成模块实现，它将零件程序、控制参数、刀具信息等读入系统，对零件程序以程序行（也称为代码段）为单位进行处理，把其中的各种零件信息，加工速度信息和其他辅助信息按照与轴组模块约定好的数据格式，通过共享内存传递给轴组模块，由其插值，并将插值信息实时送到伺服机构，驱动机床工作。

3.2　加工代码解释器结构设计

数控语言是数控系统的重要接口之一，其遵循标准由 ISO6983/IEA RS274D（G 代码）进化为 ISO14694（STEP-NC）。相比 G 代码，SETP-NC 具有巨大的优势，使数控系统具有自我规划、自我检测和自我决策的智能化行为[91]。但是 STEP-NC 的实现也是一项很艰巨的任务，目前在工业上广泛应用的数控语言仍然是 G 代码。结合实际情况，我们首先建立了针对 G 代码的任务生成模块。开放式数控系统最终应该支持 STEP-NC 标准，扩充其功能。

NC 代码解释器主要完成语法检查、译码和刀具补偿等任务。

数控程序的译码有以下两种形式：

（1）编译方式　指一次完成所有 NC 代码的译码和刀具补偿，这些工作在数控加工之前已经完成，因此和运动控制间接相关。所以在编译方式下，译码和刀具补偿属于非实时任务。

（2）解释方式　指一次只读取一条 NC 指令，然后进行刀具补偿、加减速、插值和位置控制，如果下一段译码没有完成，插值和位置控制则无法完成，所以在解释方式下，译码和刀具补偿属于实时任务。

这两种方式各有利弊，编译方式把译码和刀具补偿推到了非实时任务的范畴中，减小了实时任务的负担，但需要大容量的内存存放 NC 中间文件，在 PC 机上，随着内存容量的增大和存取速度的加快，这个问题已经得到解决；解释

方式则对计算机的实时处理能力提出了很高的要求，增加了系统资源的配置要求，但对 NC 代码的数量没有很大的限制。

经过权衡，我们采用编译型的译码方式，也就是说任务生成器是一个非实时线程，使用 Win32 API 编程。它在 Windows 环境下运行，以 COM 组件形式实现。

经过试验，编译 1000 行五轴联动直线运动的程序，需时 873ms，编译 1000 行平面圆弧运动的程序，需时 889ms，这些时间消耗都是可以接受的。

3.3 加工代码的段格式

每个 NC 代码行最多允许有 256 个字符。字符包括字母、数字、空格、Tab、回车等。其中空格和 Tab 键可以出现在程序行的任何位置，译码模块在读取时，会将它们去除掉，所以 NC 代码“G00 X6”和“G00X6”的意义是一样的。程序文件中可以有空行，译码模块也会将它们去除掉。另外，译码模块对大小写是迟钝的，所以程序 NC 代码“G00 X6”和“g00 x6”的意义是一样的。每个 NC 代码行可以包含以下四个部分。

（1）斜杠（“/”） 当一个程序行的开头有一个斜杠时，译码模块将此行忽略，甚至都不读取。

（2）行号 行号以字母“N”开头，其后跟一个整数，这个整数范围从 1 到 99999，如果超过这个范围译码模块将报错。

（3）程序本体 构成程序本体的要素为程序字。程序字由地址和后续的数值组成。例如：“X 6”，其中，“X”即是地址，而“6”就是该地址的值。地址由一个 ASCII 码表示（但不是所有的字母都有定义。例如：“W”、“V”等），如果程序行中出现没有定义的字符（例如：W、V 等），译码模块也会报错。地址规定了其后续数值的意义。根据代码中准备指令的不同，同一地址会有不同的含义，并且其后续数值的有效范围也会有所不同。超出该范围，译码模块也会报错。

（4）程序行结束符 即回车。

在以上四个部分中，第一部分是可有可无的。而其后的三个部分是程序行所必须具有的，如果缺少其中的任何一个部分译码模块都将报错。

3.4 加工代码的解释实现技术

3.4.1 译码数据处理过程

数据处理中用到的相关数据结构及双端队列等的定义如下：

1）定义 NC 代码段的插值类型：直线、圆弧等。

```
typedef enum {LINE, ARC, NURBS, DNURBS} SINGLESTEP_TYPE;
```

2）定义进给模式：快进、工进。

```
typedef enum {RAPID_MOVE, FEED_MOVE} STRAIGHTLINE_MODE;
```

3）定义线性插值的数据结构。

```
typedef struct line_struct {
  double begin_x, begin_y, begin_z, begin_a, begin_b, begin_c;
  double end_x, end_y, end_z, end_a, end_b, end_c;
  double start_rate, traverse_rate, end_rate;
  STRAIGHTLINE_MODE straightLine_mode;
  int activePlane;
} line;
```

4）译码时，对于整个 NC 代码的每一段，不管是线形，还是圆弧等类型，定义一个统一的数据结构，用于保存，以便作为同一类数据以一个数据节点保存到数据链表中去，以下正是这样的定义：

```
typedef struct singleStep_struct {
  SINGLESTEP_TYPE singleStep_type;
  double cutter_comp_radius;
  line line_struct;
  arc arc_struct;
  m_struct m_struct;
} singleStep;
```

5）以该数据结构为节点的双端队列。

```
typedef std::deque<singleStep> singleStep_deque;
```

6）具体的对象。

```
singleStep_deque singleStep_form;
```

7）刀具补偿处理后的数据结构。

```
typedef struct singleStep2_struct {
  SINGLESTEP_TYPE singleStep_type;
  line line_struct;
  arc arc_struct;
  m_struct m_struct;
} singleStep2;
```

8）以该数据结构为节点的双端队列。

```
typedef std::deque<singleStep2> singleStep_deque2;
```

9）具体的对象。

singleStep_deque2 singleStep_form2。

3.4.2 读入加工代码

将程序中的信息提取出来，存储到相应数据结构中，并检查读入过程中是否有错误。

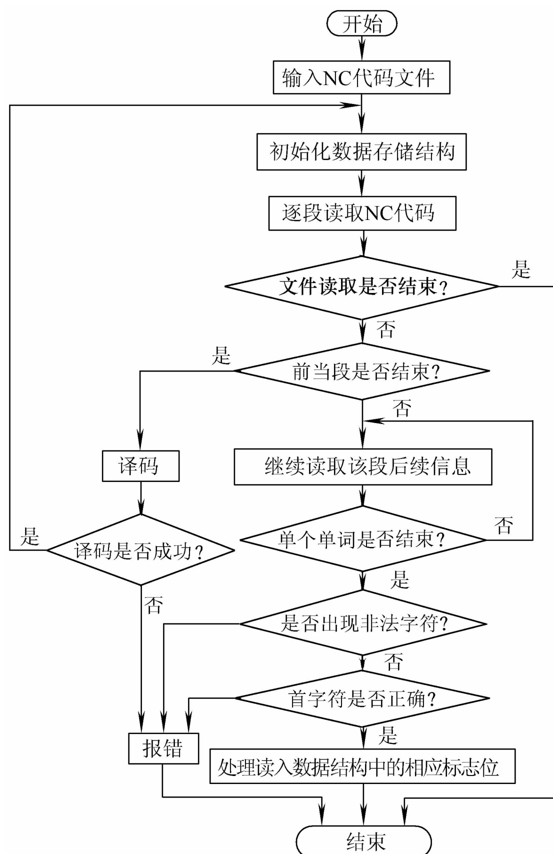


图 3-1 读入 NC 代码流程图

如图 3-1 所示，首先，逐个字符地读入 NC 代码，对构成 NC 代码的字符流进行扫描和分解，从而识别出一个个单词。这里所谓的单词是指逻辑上紧密相连的一组字符，这些字符具有集体意义，是最小的语法单位，在数控程序中称为程序字。为了有效地从源程序文本中分离出这些单词，需要先对源程序文本输入的字符串进行编辑，消除输入代码中的注释、空格、换行符以及其他一些对语法分析和数据处理无用的信息。并且，将程序字符串都转换为相应的小写

字符。之后以中间代码的形式保存有用的程序行。然后译码模块从中间代码依次读取各字符，根据地址将对应该地址的读取代码存储结构中的变量赋值为该地址的值，就完成了读取的工作。如果在此过程中调用的任何一个函数出现错误，译码模块都将返回错误代码并通过错误处理函数将该错误打印出来。

3.4.3　解释加工代码

译码模块按照一定的先后顺序处理执行位于同一个 NC 代码段的指令，这点是非常重要的，其处理顺序如图 3-2 所示。

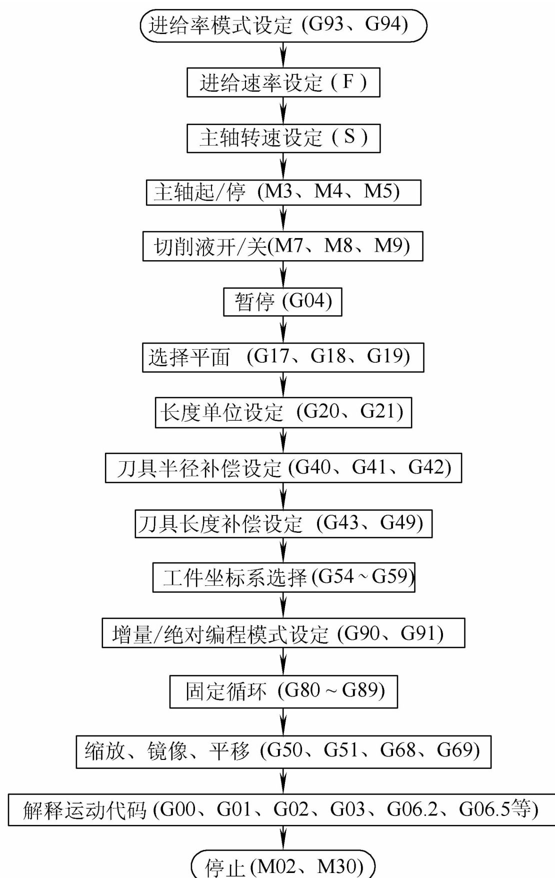


图 3-2　译码执行顺序

根据读入存储结构中的信息，将 NC 代码语句译码成执行步（Execution-Step)，并在译码过程中检查是否出现错误。

首先，检查读取代码存储结构的每一个变量值，如果该变量值有意义（译码模块将它初始化，如果在读取过程中其值被改变过，就认为有意义)，就调用

相应的函数进行处理、翻译。最后将翻译得到的 NC 代码段起始点坐标、终点坐标、速度、主轴转速等信息依次填写到双端队列中。定义如下：

typedef std:: deque < singleStep > singleStep_deque;

singleStep 的定义见 3.4.1 节，其中，line_struct 是用来存储直线运动信息的结构，arc_struct 是用来存储圆弧运动信息的结构，m_struct 是用来存储非运动信息的结构。整个程序译码完成后，根据双端队列中的 NC 代码信息进行刀具补偿，将刀具补偿后的信息也放到一个双端队列中去，然后通过共享内存传递到轴组模块。

3.5 出错处理机制

由于手工或用 CAD/CAM 技术编写的 NC 代码难以避免会出现这样那样的错误，例如：未定义的输入字符、缺少指令或参数值、多余指令或参数值、参数值越界、不合法的字符组合等。如果不进行语法检查就直接插值执行，可能会造成重大事故。为保证控制软件的正常运行，译码模块在读入和编译期间总是检查各种情况。但是译码模块对机床的错误无能为力，例如：机床在某个轴方向超出行程、光纤环错误等，这些问题将由该开放式数控系统 Soft-PLC 模块来保证。

3.5.1 常见出错分析

系统处理的常见错误有以下几种：

1）文件操作失败，包括打开失败，打开空文件等。

2）未定义字符。

3）同一地址重入问题。对于大部分地址其参数值不得重入（G、M 除外），即同一程序代码行中同一地址符出现的次数不得多于一次。

4）同组 G 代码或 M 代码重入问题。同一程序行中不允许位于同一组的多于一个的 G 代码出现。

5）特定语句之间互相约束问题。例如：设定坐标指令不能在刀具补偿运行中使用等。

6）其他错误。除了上述 5 种以外还有很多种错误，例如：在刀具补偿过程中发现刀具半径大于要加工的圆弧半径等。

3.5.2 错误处理方法

译码模块在读入与编译阶段时刻检查错误。

所有读入和编译级的函数都有返回值。每个函数都通过判断它调用的下层

函数的返回值来判断下层函数在调用期间是否发生错误。通过返回调用函数的方法一层一层地返回错误的特定值，最后由报错函数统一处理，而不是在错误发生的时候就直接跳出，并报错。

如果没有发生以上情况中的任何一种，则返回错误的特定值。本译码模块定义了四个宏，由它们来检查错误是否发生，一旦发生错误，准确标识该错误，给它一个特定值，并保存发生错误的程序代码行，以便使用者能够从错误中找到发生错误的地点和原因。这个特定值就包含着这个错误的信息。同时建立了一个错误信息数组，错误处理函数通过将错误返回值与相应错误信息数组中的文档相匹配，将错误文档和出错的代码行打印在指定的文件中，同时，在人机界面上将其显示出来。

3.6　任务生成指令及其功能

在加工代码中一般有准备功能指令（G 指令）、辅助功能指令（M 指令）、进给速度指令（F 指令）、主轴转速指令（S 指令）、刀具选择指令（T 指令）等。

3.6.1　指令分组原则

在所有指令中有的引起机床状态的改变或运动的形成，而这些状态和运动指令没有被显式取消之前将一直维持下去，将它们称为模态代码。而另外的就是非模态代码，在任务生成模块中将所有可以使用的 G 代码按作用和是否为模态分为 15 组，见表 3-1。

表 3-1　系统中已实现的 G 代码

G 代码	组别	功能	G 代码	组别	功能
G0	1	快进	G20	6	英制长度单位
G1	1	直线进给	G21	6	公制长度单位
G2	1	顺时针圆弧进给	G4	0	暂停
G3	1	逆时针圆弧进给	G10	0	坐标系原点设定
G06. 2	1	3 轴联动 NURBS 插值	G28	0	回原点
G06. 5	1	5 轴联动 NURBS 插值	G30	0	回参考点
G17	2	选择 XY 平面	G40	7	取消刀具半径补偿
G18	2	选择 YZ 平面	G41	7	左侧刀具半径补偿
G19	2	选择 XZ 平面	G42	7	右侧刀具半径补偿

（续）

G 代码	组别	功能	G 代码	组别	功能
G43	8	刀具长度补偿	G81	1	钻削循环
G49	8	取消刀具长度补偿	G82	1	钻、镗阶梯孔循环
G50	14	缩放、镜像开始	G83	1	深孔加工循环
G51	14	缩放、镜像结束	G85	1	镗孔循环
G53	3	选择机床坐标系	G86	1	镗孔循环
G54	12	选择工件第一坐标系	G88	1	镗孔循环
G55	12	选择工件第二坐标系	G89	1	镗孔循环
G56	12	选择工件第三坐标系	G90	3	绝对坐标模式
G57	12	工件第四坐标系	G91	3	相对坐标模式
G58	12	工件第五坐标系	G92	0	坐标系原点偏置
G59	12	工件第六坐标系	G93	5	时间倒数进给率
G68	14	坐标旋转开始	G94	5	每分钟进给率
G69	14	坐标旋转结束	G98	10	返回初始平面
G80	1	取消固定循环	G99	10	返回参考平面

其中，第 0 组为非模态代码，其他组都是模态代码。在同一程序行中每个组只能有一个代码出现，否则，任务生成模块报错。组与组之间代码不冲突（除了第 1 组和第 0 组），所以在同一代码行中可以有 14 个 G 代码，只要它们来自不同的组就可以。

类似的将已实现的辅助功能 M 代码进行分组，见表 3-2。

表 3-2　系统中已实现的 M 代码

M 代码	功能	M 代码	功能
M2	程序停止	M8	雾状切削液开
M3	主轴正转	M9	切削液关
M4	主轴反转	M30	程序停止
M5	主轴停止	M48	速度倍率允许
M7	液状切削液开	M49	速度倍率取消

在 G 和 M 代码组中有一些代码是机床默认的代码，在控制器开机或重启时它自动生效。它们是：G17（XY 平面）、G90（绝对模式）、G94（每分钟进给

率）、G20（公制单位）、G98（返回参考平面）、M5（主轴关）、M9（切削液关）、M48（速度和主轴转速倍率开）。

3.6.2　固定循环指令

准备功能指令的第 1 组与第 0 组中的 G28、G30 都是产生机床刀具运动的指令。其中的固定循环指令是研究的重点。固定循环用于钻削、镗削通孔、阶梯孔，在一个程序行中用一个固定循环指令完成快进到参考平面、工进钻削、孔底暂停、快回参考平面（或初始平面）等多个动作。与增量模式（G91）配合使用还可以完成用一行指令加工多个孔的功能，大大减少了用户编制零件程序代码的工作量，方便了用户的使用。

每个固定循环指令都有相同的一系列动作：快进到参考平面、快退到参考平面。也有其自己特定的一些动作，比如 G82 指令需要在孔底暂留一段时间（以便断屑）。故而在本译码模块中将相同的动作提取出来，统一处理，同时为不同的指令编写各自函数，实现其特定动作。下面这段代码所定义的宏指令就是固定循环的上层调用函数，每个指令都调用它，由它来完成相同的动作，而这个函数又会调用对应于不同指令的相应下层函数，来完成其特定动作。

```
#define CYCLE_MACRO (call) for (repeat SET_TO block->1_number;    \
    repeat >0; repeat--) {                                         \
  aa SET_TO (aa + aa_increment);                                   \
  bb SET_TO (bb + bb_increment);                                   \
  cycle_traverse (plane, aa, bb, old_cc);                          \
  if (old_cc ISNT r) cycle_traverse (plane, aa, bb, r);            \
  CHP (call);                                                      \
  old_cc SET_TO clear_cc;}
```

例如：“G82 X10 Y20 Z-10 R5 P8”，假设现在机床刀具位置为 X0 Y0 Z10 则译码模块在读取该行程序后，首先调用 CYCLE_MACRO ( )，由它来完成快进到 X10 Y20 Z5 的动作，然后该函数继续调用 convert_cycle_g82 ( ) 函数，由它来完成工进到孔底 Z-10，并在孔底暂停 8s，然后函数返回 CYCLE_MACRO ( ) 函数继续完成快退到参考平面（或初始平面）的动作。这样整个孔加工循环完成。

对于程序“G91 G81 G98 X4 Y5 Z-0.6 R1.8 L3”，假定现在机床刀具点在机床坐标系中的位置为 X1 Y2 Z3。则机床将完成如下动作：

快进到点（5，7，4.8）；

工进到孔底（5，7，4.2）；

快退到初始平面（5，7，4.8）；

然后加工第二个孔；

快进到点（9，12，4.8）；

工进到孔底（9，12，4.2）；

快退到初始平面（9，12，4.8）；

类似的加工第三个孔；

快进到点（13，17，4.8）；

工进到孔底（13，17，4.2）；

快退到初始平面（13，17，4.8）。

这样就用一行程序完成了三个孔的加工。

在固定循环中，加工完孔后刀具到底是快退到参考平面还是初始平面是由 G 代码 G98（初始平面）、G99（参考平面）决定的，其中 G98 为机床默认模式，并且两者都为模态指令。

3.6.3 坐标系建立方式

在本系统中与机床有关的坐标系包括一个机床绝对坐标系和六个工件编程坐标系。

可以用指令“G10 L2 Pn”（n 为工件坐标系）以及相应坐标值来设定工件坐标系相对于绝对坐标系的原点坐标，这样工件坐标系就被唯一确定下来了。例如：“G10 L2 P1 X10 Y20 Z30”，则第一个工件坐标系的原点就将被设定在机床绝对坐标系的 X10 Y20 Z30 位置。用户可以通过指令 G54 ~ G59 来选择使用这六个工件坐标系，一经选择，其后程序行坐标系就将变成所选工件坐标系，一直持续到取消此坐标系为止。当选择了某一坐标系后，还可以通过 G92 指令对该坐标系再进行偏移，一旦使用该指令进行偏移，偏移量将对六个坐标系都有效，直到取消该指令。在系统初始化过程中，选择哪个坐标系可通过参数文件中的第 5220 号参数选定。

如果在某一特定程序行需要使用绝对机床坐标，可以使用 G53 指令，该指令为非模态，只对该行有效。

3.6.4 系统参数设定

每台机床都有其固定的参数设置，大部分都有其固定用途。参数必须全时间存在，即使是机床关机的情况下。在本软件控制系统中，采用文件的形式完成此功能，在系统启动的时候，首先打开参数文件并将参数读到相应位置，在系统运行过程中，涉及参数修改时，直接将系统相应位置参数修改。参数文件见表 3-3。

参数从 1 到 5400，每个文件中除了必须包含上述必需参数外，其他参数可

以根据需要适当添加，但是参数必须按照升序排列。当必需参数缺失或者参数没有按照升序排列时，任务生成模块将报错。

表 3-3　参数文件

参数代码	参数值	注释	参数代码	参数值	注释
5161	0.0	G28 原点 *X*	5211	0.0	G92 偏置 *X*
5162	0.0	G28 原点 *Y*	5212	0.0	G92 偏置 *Y*
5163	0.0	G28 原点 *Z*	5213	0.0	G92 偏置 *Z*
5164	0.0	G28 原点 *A*	5214	0.0	G92 偏置 *A*
5165	0.0	G28 原点 *B*	5215	0.0	G92 偏置 *B*
5166	0.0	G28 原点 *C*	5216	0.0	G92 偏置 *C*
5181	0.0	G30 参考点 *X*	5220	1.0	坐标系选择
5182	0.0	G30 参考点 *Y*	5221	0.0	坐标系 1*X*
5183	0.0	G30 参考点 *Z*	5222	0.0	坐标系 1*Y*
5184	0.0	G30 参考点 *A*	5223	0.0	坐标系 1*Z*
5185	0.0	G30 参考点 *B*	5224	0.0	坐标系 1*A*
5186	0.0	G30 参考点 *C*	5225	0.0	坐标系 1*B*
			5226	0.0	坐标系 1*C*

3.6.5　可扩展宏的定义

在一个程序文件中如果存在一个参数被多处数次使用，那么，用户可以将它定义为宏，这样可以统一管理它，只要在宏定义处将它的值进行修改，那么，在程序中所有出现的地方其值都将跟着变化。宏的定义是以一个“#”符号紧跟着一个整数值，这个整数值就是该宏的序号，这个序号的范围可以是 1 ~ 5399，也就是说在一个程序文件中用户可以自定义 5399 个宏，如果超出这个范围，译码模块将报错。宏符号“#”的优先级高于其他一切表达式操作符的值（当然低于括号的优先级）。所以，程序“#1 + 2”表示 1 号宏的值加上 2，而不是 3 号宏。另外，程序“##1”表示 1 号宏所存储的值（假定为 2）为序号的宏（也就是 2 号宏）。用户自定义宏的赋值与其他变量赋值是一样的，如“#2 = 15”。但是用户自定宏赋值以后必须等到赋值行运行完以后才有效，例如：#3 在程序文件中已经被赋值为 15，在其后的某一行出现“#3 = 10 G01 X#3”，那么，译码模块首先让机床运动到 *X* 坐标为 15 的位置后，再将 3 号宏值修改为 10，在其后再

用到 3 号宏，就都将按照 10 来运行了。

3.6.6 运算符与函数定义

本软件所支持的表达式运算符与函数见表 3-4。

表 3-4 运算符与函数

运算符或函数	说明	运算符或函数	说明
+	加	*	乘
-	减	/	除
SIN（）	正弦	SQRT（）	平方根
COS（）	余弦	ABS（）	绝对值
TAN（）	正切	ROUND（）	四舍五入取整
ASIN（）	反正弦	MOD（）	模除
ACOS（）	反余弦	EXP（）	指数运算
ATAN（）	反正切	OR	或
AND	与	XOR	异或

用户在编写 NC 代码的时候，在坐标值和参数值的位置，可以直接输入表达式，而不必计算出其值，将计算的任务留给译码模块来完成，例如：“sin1.1”，这样的表达式是不好计算的（如果没有计算器），如果用户直接将此表达式写在程序中，让译码模块来计算它，用户的工作量将大大减少。

在本控制器中规定所有表达式都以“[”开始，以“]”结束。在表达式中可以出现数字、用户自定义宏、数学操作符、逻辑操作符等。如果出现未定义符号，译码模块将报错。在译码模块读取程序的时候表达式就将被计算出来，它所在的位置由计算出来的数值来代替。

3.6.7 注释代码

为了让用户在编写 NC 代码时，将每个程序行的意义写下来，以便以后使用或修改时一目了然，本译码模块支持用户在程序文件中加入注释的功能，当然，译码模块对这些注释并不执行，只是将它们忽略掉。所以注释都写在“()”中，例如：“G40（stop compensation)”。

3.6.8 子程序调用

零件代码程序可以分为主程序和子程序，数控系统按照主程序指令的顺序

操作。程序中有固定顺序和可重复执行的一个部分，可将其作为子程序存放，使整个程序简单化，也可以使编程效率提高。主程序可以多次调用同一个子程序，也可以在不同地方多次调用不同的子程序。子程序以字母“O”开头，以指令“M99”结束。主程序以指令“M98”调用子程序，例如：程序段“M98 P02 L3”的意思就是调用第二号子程序，并连续调用三次。

3.7　刀具补偿机理

本课题涉及铣削加工和钻削加工。所以本书只论述关于这两种加工的刀具补偿，另外，本书进行的刀具补偿指的是平面刀具补偿。同时，由于刀具长度补偿比较简单，在这里将不单独进行讨论。

3.7.1　概述

在轮廓加工中，所用的刀具半径都有一定的大小，如数控切割机床的切割火焰半径、等离子弧半径、铣床上的铣刀半径等。刀具中心的运动轨迹与所需加工零件的实际轨迹并不相同，而是刀具中心相对于零件的实际轨迹有一个偏移，上述偏移，习惯上称为刀具的半径补偿。

值得指出的是刀具补偿通常不是由零件程序编制人员完成的。在现场，程序编制人员只是根据零件加工轮廓编制程序。

实际中，当程序编制人员按零件轮廓编制程序时，各程序段之间是连续过渡的，没有间断点，也没有重合段。但在进行了刀具的半径补偿（原来的 B 刀具半径补偿）后，两个程序段之间的刀具中心轨迹就可能出现交叉与间断。对于这种只有 B 刀具半径补偿的 CNC 系统，程序编制人员必须事先估计出在进行刀具半径补偿后可能出现的断点与交叉点，进行人为处理。如在交叉处增设一半径比刀具半径大的过渡圆弧，在断点处加一半径为刀具半径的过渡圆弧。很明显，这种方法还是不能完全按零件轮廓编制加工程序。

随着 CNC 技术的发展，系统工作方式、运算速度及存储容量都有了很大的改进与增加，采用直线或圆弧过渡，直接求出刀具中心轨迹交点的刀具半径补偿方法已经能够实现。这种刀具半径补偿方法叫 C 功能刀具半径补偿法。C 功能刀具半径补偿就是零件加工程序完全按零件轮廓编制，由计算机自动控制刀具的中心轨迹，不会出现交叉与间断的补偿法[92~95]。

3.7.2　数据存储管理模式

在 C 功能刀具半径补偿方法中，计算完本段轨迹后即提前将下一段 NC 代码读入，再根据它们之间的转接情况，对本段轨迹进行修正，最终得到正确的刀

具中心轨迹。

图 3-3 为刀具补偿所用的数据存储管理方式。计算开始时，译码后的第一个运动段代码信息首先被从双端队列中读入到第一 NC 代码段寄存器 AR 中，接着第二个运动段 NC 代码被读入到 BR 中，然后，对第一、第二两段 NC 代码轨迹的转接方式进行判别，根据转接类型，对 AR 中的第一段编程轨迹进行修正。修正后，将第一段的编程轨迹由 AR 送至 CR，然后，将这个带有刀具补偿信息的运动段 NC 代码送到存储刀具补偿后程序的输出双端队列中去。接着将第二段的编程轨迹由 BR 送至 AR。依次执行。

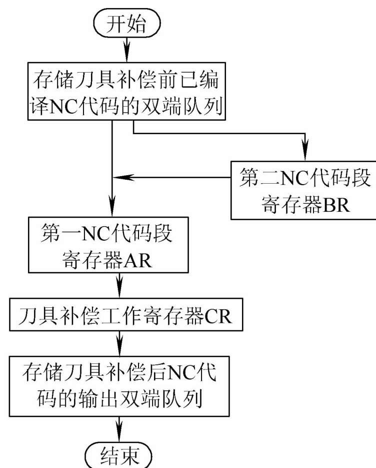


图 3-3 C 功能刀具半径补偿数据存储管理

3.7.3 刀具补偿实现机理

刀具补偿信息的加入需要等 NC 代码文件被译码模块读入并作相应处理后实现。处理后的 NC 代码信息按每行一个节点存入双端队列中。

如图 3-4 所示，首先，进入功能入口函数 Rs274ngc_cutter_comp ( )，然后依次调用相关函数，进入 Cutter_comp_main ( ) 后首先从双端队列中依次提取节点，判断是否为运动段（需要进行插值的 NC 代码段），如果不是，则将此非运动段 NC 代码直接送入作为输出缓冲器的第二个双端队列中去，并继续向下查找，直到找到运动段 NC 代码或最后一个程序段节点为止。如果找到运动段，就继续判断是否为最后一个 NC 代码段，如果是则也直接送到第二个双端队列中，如果不是，则送到寄存器 AR 中去。接着，从第一个双端队列中从上一运动段 NC 代码节点向后再顺序查找运动段 NC 代码节点，不是运动

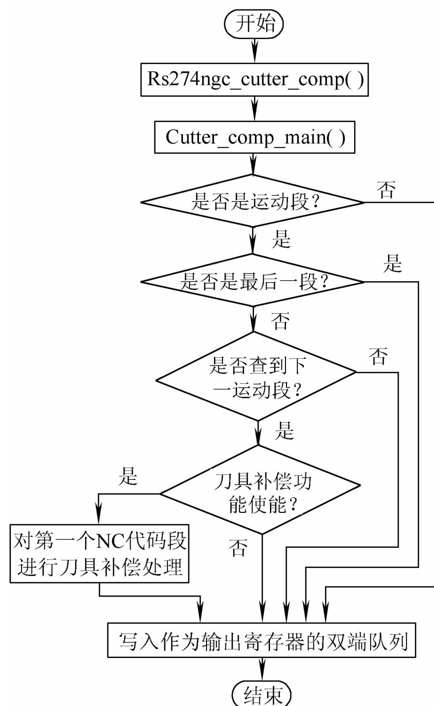


图 3-4 C 功能刀具半径补偿预处理

段 NC 代码的节点仍直接送到第二个双端队列，直到找到又一个 NC 代码段节点，将这个节点放到寄存器 BR 中去。此时，就可以对第一个运动段 NC 代码进行刀具补偿预处理。首先，判断两个运动段是否处于刀具补偿使能的状态，如果两个运动段刀具补偿都没有使能，就不进行刀具补偿处理，而直接送到第二个双端队列中去。

如果有一个或两个运动段处于刀具补偿开的状态，就要对第一个 NC 代码段进行刀具补偿处理。这时通过入口函数 Actual_comp（）进入刀具补偿处理部分，如图 3-5 所示。首先判断两个 NC 代码段的类型，即是直线段还是圆弧段，按照判断结果分别进入不同分支结构继续执行。以两个 NC 代码段都是直线段为例说明，判断两 NC 代码段之间的转接类型（缩短型、伸长型或插入型），根据判断结果进入相应分支，继续判断刀具补偿运行状态（刀具补偿建立、运行中或撤销）并进入相应分支。这时就可以调用相应的最底层函数，对原程序段的坐标进行修正，使之成为刀具中心点坐标，必要时可以在两段 NC 代码之间插入一段或几段直线段以完成两个 NC 代码段之间的转接。最后，将刀具补偿后的 NC 代码段节点送到第二个双端队列中。

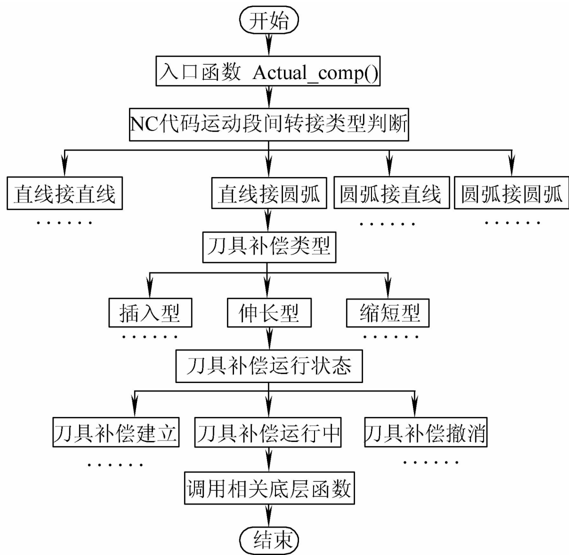


图 3-5　C 功能刀具半径补偿程序流程图

限于篇幅，刀具半径补偿算法这里不再赘述。

3.8　小结

本章给出了开放式软数控系统中 NC 代码解释器的实现方法。系统采用编译

方式，将 NC 代码文件读入，进行解释及刀具补偿，最后形成执行步，将所有执行步一起送到轴组模块进行插值，进而控制机床运动。充分利用了计算机大存储容量和快速的存储速度，同时减轻了实时任务部分的工作量。NC 代码解释器采用模块化结构，为系统跨平台移植保留了很大的空间。同时在内部规划上采用层级式函数文件结构，当用户需要增加功能或将译码模块移植到其他种类机床时，只要对其中相应文件进行修改就可以了。

本解释器实现了 C 型刀具补偿，并实现了固定循环指令、平面选择、坐标系选择、旋转、缩放、镜像、子程序调用等功能。丰富了指令系统，使之基本达到了普通数控系统水平。

第 4 章 Soft-PLC 技术原理及实现

本章将集中论述多轴 Open-CNC 数字制造系统中 Soft-PLC（Programmable Logical Controller，可编程序逻辑控制器）系统的实现方法。

随着计算机技术的迅猛发展和广泛应用，传统 PLC（可编程序控制器）充分暴露出缺乏通用性和兼容性的缺点，而且新的计算机软、硬件技术和网络技术不能被 PLC 系统及时利用，这既不利于 PLC 产品的更新换代，也不利于降低成本[96,97]。因此，在国际标准 IEC61131-3 的基础上，人们发展了一种以通用操作系统和 IPC 为软硬件平台，用软件实现传统硬件 PLC 的控制功能的新的 PLC 技术，即 Soft-PLC 技术。与传统 PLC 相比，Soft-PLC 具有开放的体系结构，强大的网络通信能力和更强的数据处理能力，能更好地满足现代工业自动化的要求。

在数控系统中，PLC 在处理开关量辅助控制问题时起着重要作用，它主要负责 NC 侧和机床侧的逻辑信号处理。在 NC 侧，CNC 向 PLC 发送 M、S、T 等辅助功能代码信息；PLC 将 M、S、T 命令的应答信号回送给 CNC，并且控制 CNC 设置各坐标的机床基准点。在机床侧，PLC 向机床传送控制机床执行的信号，机床将其操作面板上开关、按钮信号等传送给 PLC。

4.1 概述

4.1.1 以软件实现传统 PLC 功能

PLC 是工业控制用的专用设备，它的硬件组成与计算机应用系统的硬件组成类似。传统 PLC 主要由以下几部分组成[98]：CPU，存储器，控制用 I/O 接口，外设用 I/O 接口，扩展用 I/O 接口，电源。传统 PLC 可靠性高、抗干扰能力强，适用于恶劣的控制现场，在工业控制领域得到广泛应用。但各厂家产品体系结构是封闭的，其软硬件平台各不相同，支持 PLC 编程的语言及指令系统各异。用户选择一种 PLC 产品后，必须选择与其相应的控制规程，学习特定的编程语言，不利于终端用户功能的扩展。而要想将不同厂商的产品应用于同一个工业控制网络尤其困难。

随着计算机技术的迅速发展，人们逐渐意识到传统 PLC 的通用性和兼容性不好以及极其昂贵的价格正制约着 PLC 的快速发展。同时工业自动化控制领域

的国际标准 IEC61131-3 的推出和实施，打破了以往各个 PLC 生产商产品相互不兼容的局限性。依据这个国际标准，充分利用工业控制计算机 IPC 或嵌入式计算机 EPC 的软硬件资源，全软件形式实现传统 PLC 功能的新技术——Soft-PLC（SoftPLC）技术应运而生[99,100]。

Soft-PLC 是一种基于 IPC 或 EPC（嵌入式 PC）的软逻辑控制软件，与传统 PLC 一样，主要用于工业控制领域中的开关逻辑控制、顺序控制、定时控制、计数控制、闭环过程控制（PID 控制、PWM 控制）等。Soft-PLC 综合了传统 PLC 的一些特色和计算机多任务处理技术、强数据处理能力、强大的网络通信功能以及具有友好的人机界面等特点，形成新的具有开放体系结构的工业控制器。工业领域已经开始使用 Soft-PLC 产品，而且 Soft-PLC 的市场需求量也在不断地增长。市面上典型的 Soft-PLC 应用产品有：SOFTPLC 公司的 SoftPLC，SIEMENS 公司的 SIMATIC WinAC 以及 CJ International 公司的 ISaGRAF 等[101]。

Soft-PLC 是一个具有高附加值和高技术含量的产品，符合 IEC61131-3 标准的 Soft-PLC 相比于传统 PLC，存在以下优势：具备柔性扩展能力，用户可以选择不同厂家生产的各种硬件产品，充分利用每个软件的最佳特性，组成最佳的控制系统；产品开发周期短，编程、调试和维护方便；产品通用性和兼容性高，可以运行在不同的 IPC 或 EPC 上，并且共享 PC 机上的软硬件资源，从而使 Soft-PLC 的生产和运行成本较低，体现 Soft-PLC 性能价格比要高于传统 PLC 的特点。

4.1.2 IEC61131-3 软件系统

IEC61131-3 是首个为工业控制系统提供标准化编程语言的国际标准[102]，该标准针对工业控制系统所阐述的软件设计概念、模型等适应了当今世界软件、工业控制系统的发展方向，是一种非常先进的设计技术，不但极大地推动了工业控制系统的软件设计的进步，而且它的许多概念还对现场总线设备的软件设计产生了很大影响。符合 IEC61131-3 的软件系统是一个结构完美、可重复使用、可维护的工业控制系统软件，不但能被应用在 PLC，而且还能被应用在控制工业及制造过程的一切软件中，是一种先进的工业控制编程系统。限于篇幅，这里不再对 IEC61131-3 作详细介绍。

IEC61131-3 标准包括两个部分：程序和公共元素。公共元素定义了编程系统中需要的数据类型。程序部分主要描述了两个重要的模型：软件模型和通信模型，是本书参考 IEC61131-3 标准的主要部分。

4.2 Soft-PLC 系统结构

本系统将 PLC 的控制功能封装在软件内，运行于 PC 的环境中，即提供了

PLC 的相同功能，也具备了 PC 机的各种优点：开放的体系结构，可以支持多种硬件和编程语言，以及可灵活扩展系统的功能；能够充分利用硬件资源；友好的人机界面，便于操作；更强的数据处理能力和控制算法；节约成本。

Soft-PLC 系统由编程开发系统（简称开发系统）和运行系统组成[103]，是相互独立、又联系密切的两个应用程序，并可分别单独运行。系统的运行过程是先用 PLC 开发系统编辑开发 PLC 控制程序，生成目标代码，然后由 PLC 运行系统运行这个目标代码，来实现对系统的控制。

Soft-PLC 开发系统是带有调试和编译功能的 PLC 编程器，它具备以下几个功能：①绘制和编辑 PLC 程序；②编译用标准语言编写的 PLC 程序，能够查询并显示程序中的错误，经用户改正后生成目标代码，传入运行系统执行；③仿真功能，实现离线仿真和在线修改；④强大的网络功能，支持 TCP/IP 协议的网络通信，通过网络实现远程监控。

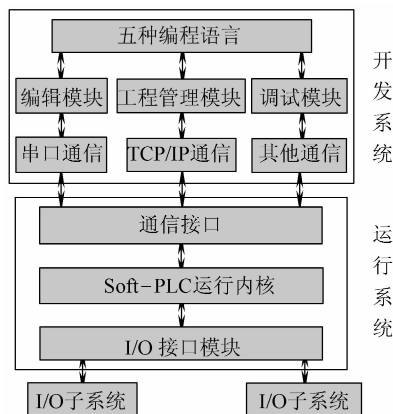


图 4-1　Soft-PLC 系统结构

Soft-PLC 运行系统是 Soft-PLC 的核心，完成输入处理、程序执行、输出处理等工作。由 I/O 接口、通信接口、系统管理器、错误管理器、调试内核和编译器组成。Soft-PLC 运行系统将开发系统生成的程序编译链接成可执行文件运行，用输出结果反映 I/O 状态的改变。

开发系统和运行系统组成是相互独立、密切相关的两个应用程序，并可分别单独运行。Soft-PLC 系统的整体框图如图 4-1 所示。

4.3　Soft-PLC 编程开发系统设计

根据 Soft-PLC 的基本原理和开发系统应实现的基本功能，将系统分为编辑和编译两大模块。

根据 IEC61131-3 对 PLC 编程语言的规定和五种语言的各自特点，目前选择了图形模式语言梯形图 LD 和文本模式语言指令表 IL 作为本系统的编程语言。梯形图是目前应用最广泛的直观的编程语言；指令表语言不但简单易学，而且非常容易实现，且 IEC61131-3 的其他语言如功能块图、结构化文本、梯形图等都可以转换为指令表语言。由于系统的开放性，可以根据以后的需要，将其他三种编程语言加载到系统中来。

4.3.1 编程开发系统中编辑器的实现

编辑模块分为三个部分：梯形图编辑器、指令表编辑器以及梯形图到指令表的转换模块。可供用户选择的两种 PLC 编程方法是梯形图编程和指令表语言，并由转换模块实现从梯形图到指令表的转换。

（1）梯形图编辑器 梯形图是一种图形类语言，因此，语言的组成部分是表示各个触点、线圈、功能的符号资源，以及记录每一个符号命令的内部参数等。根据梯形图的图形表示特点，将各符号、功能块通过位图的形式表示，并一一显示在梯形图编辑区内，完成梯形图的绘制。通过对梯形图符号的分析，将其归纳为各种符号的基本图库。

在本系统中，梯形图编辑器实现的主要功能包括：

1）绘制梯形图。依照 IEC61131-3 中对梯形图符号的规定，按照用户要求，将梯形图显示在梯形图编辑区内。

2）绘图资源的保存。供程序运行时调用。

3）编辑功能。包括插入、删除图形；插入、删除行；回车自动产生左母线；查找功能等。

4）设置快捷键、工具栏、状态栏。快捷键、工具栏方便用户使用，状态栏显示当前状态等。

5）文件的保存和读取。

（2）指令表编辑器 指令表语言是一种汇编语言风格的编程语言，由于指令表程序的高效、执行速度快而为软件工程师或高级专业工程师所喜欢。指令表语言在五种编程语言中的地位，就如同计算机汇编语言在程序设计语言中的地位一样，是一种底层的编程语言，在 IEC61131-3 软件结构中的作用不可替代。因此，指令表语言不仅仅是五种编程语言中的一种，在软件结构内部，它还起到将其他文本语言和图形语言编译生成或相互转换的公共中间语言的作用。

指令列表程序编辑器是一个文本编辑器，所有的逻辑和运算都使用指令和操作数的方式输入，根据指令所完成的功能和涉及操作数中的软元件，完成软元件值的读取、逻辑处理和软元件值写入。

（3）转换模块的实现 在此模块中，将已经绘制的梯形图转换成相同逻辑功能的指令表形式表示，并显示在指令表编辑视窗中。

4.3.2 开发系统中编译器的实现

在本控制器的研究中，开发系统的主要功能是编辑、编译 PLC 编程语言，最终将生成的目标代码传递到下位机运行。因此，编译部分是开发系统的关键。

IEC61131-3 制定了五种编程语言方法[104]，其中三种是图形化语言：顺序功

能图（SFC），梯形图（LD），功能块图（FBD）；两种是文本化语言：指令表（IL），结构文本（ST）。产品不要求能够全部运行五种语言，可以只运行其中几种，但要遵守其标准。

其中，指令表语言类似于汇编语言，它常用于用户自行编制一些没有标准功能块的特殊算法，具有很大的灵活性，较高的透明度，其他各种语言均可以转化成指令表语言。且语言表达形式简单，算法明了，易于编译。因此在本系统的研究中，选择将其他四种语言转换为指令表语言，再对其进行分析和编译。

指令表是面向行的编程语言，一条指令对应 PLC 控制器可执行的一项命令，其指令结构如图 4-2 所示。对应的实例如下：

001：OT Y0.0（＊output＊）

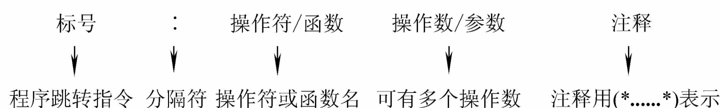


图 4-2　指令表结构图

语句表指令主要由操作码和操作数组成，有些操作码可带多个操作数，这时各个操作数用逗号分开。指令前加标号，后面跟冒号，在操作数之后可加注释。

（1）编译指令表语言的步骤　本系统的编译程序是将按照指令表语言规则编写的源程序转换成等价的能在运行系统上执行的目标代码的程序。编译程序是一个高度复杂的程序，其内部结构和组织方式具有多种形式。主要工作可分为两部分：前端与后端。

前端完成分析，可分为词法分析、语法分析和语义分析[105]。

1）词法分析。指令表源程序可以被简单地看成一个多行的字符串。词法分析器逐一从前往后、从左到右地读入字符，按照源语言的词法规则，拼写成一个一个的单词（token）保存。编译程序把单词作为源程序的最小单位，等待语法分析。

2）语法分析。语法分析器读入单词，将他们组合成按指令表语法格式规定的词组。在语法分析中，如果源程序没有语法错误，就可以正确画出其分析树。否则，指出语法错误，给出相关的诊断信息。

3）语义分析。语义分析阶段主要检查源程序是否包含语义等逻辑错误，并收集类型信息以供后面的代码生成阶段使用。只有语法和语义正确的源程序才能被翻译成正确的目标代码。

后端完成综合，包括目标代码生成与代码优化。经过后端处理，生成质量高、运行速度快的可在目标机上运行的代码。其中，前端部分基本与所属机器无关，而后端与目标机密切相关，也就是 Soft-PLC 的运行系统。

以上几部分有机地结合起来，完成对指令表的编译。编译过程如图 4-3 所示。

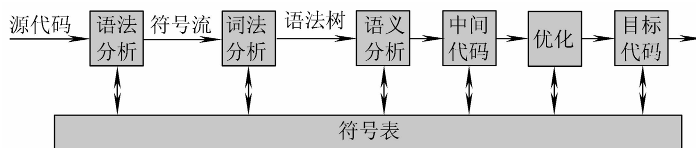


图 4-3 编译过程框图

（2）用 Lex 生成词法分析器 Lex 工具是一种词法分析程序生成器，它将一个包含了正则表达式以及每一个表达式被匹配时所采取的动作的文本文件作为其输入的程序。它可以根据词法规则的要求来生成单词识别程序，由该程序识别出输入文本，即指令表编写程序中的各个单词。Lex 程序由说明部分、规则部分及用户子程序部分组成。其中规则部分是必需的，说明部分和用户子程序部分可以任选。

1）说明部分。说明部分包含一些正则表达式（regular expression）的宏。这部分也可以包含一些初始化代码，该代码标记在"% {"和"% } "之间，如图 4-4a 所示。前面位于分隔符% {和% } 之间的部分将直接插入到 Lex 产生的 C 代码之中。正则定义部分对正则表达式命名。正则表达式命名后面跟随的就是按照指令表词法规则所定义的正则表达式。标号 line 表示一个整数，注释 comment 表示由（ * 和 * ）包围的除了 * 之外的任意表达式，whitespace 表示一个或多个空格，variable 表示的正则表达式规定了操作数的格式等。这些正则表达式包括了所有指令表语言中可能出现的符号，也就是说，指令表语言被读取的时候，每一个字符都能够被与正则表达式所匹配的规则识别出来，生成单词并保存。

2）规则部分。规则部分起始于"% % "符号，终止于"% % "符号，如图 4-4b 所示。这部分是程序的核心部分，也是必须规定的部分。规则部分由模式和动作两部分组成。模式部分可以由任意的指令表的正则表达式组成，动作部分是由 C 语言语句组成，这些语句用来对所匹配的模式进行相应处理。处理过程中，Lex 将识别出来的单词存放在 yytext [ ] 字符数据中，因此该数组的内容就代表了所识别出来的指令表程序中的单词内容。当程序读取了一个整数，即与 line 正则表达式相匹配的时候，就会执行后面的动作命令，并返回一个值 LINE。处理空格（whitespace）的动作为空，即无操作。最后一行“.”表示 Lex 如果输入不能满足上面所有的规则（除了“\ n”以外的所有字符），则返回一个错误记号。

3）用户子程序部分。用户子程序部分可以包含用 C 语言编写的子程序，用来添加在规则部分被调用而未在其他部分定义的辅助程序。如果 Lex 程序独自运行，这一部分还应添加一个主程序。本书程序中没有写这一部分。

说明部分

```
% {
    #include <stdio.h>
    #include "y.tab.h"
% }
digit   [0-9]
letter  [A-Z]
line  {digit}+
newline   [\n]
whitespace   [\t]+
variable
{letter}+{digit}". "{digit}
comment  ("(*"[^*]*")")
```

a)

规则部分

```
%%
{line} {printf("Line:%s\n", yytext);
       return  LINE; }
......
"OT"  {printf("Out:%s\n", yytext);
       return  OUT; }
      {printf("Othersymbol:%c n", yytext[0]);
       return yytext[0]; }
      {whitespace} {; }
%%
```

b)

图 4-4　词法分析器示例

a）说明部分　b）规则部分

（3）用 Yacc 生成语法分析器　Yacc（Yet Another Compiler—Compiler）是一种工具，将任何一种编程语言的所有语法翻译成针对此种语言的 Yacc 语法解析器，由该解析器完成对相应语言中语句的语法分析工作。按照惯例，Yacc 文件有 .y 后缀。

在使用 Yacc 工具前，必须首先编写 Yacc 程序，因为有关指令表语法分析程序是根据 Yacc 程序生成的。为了使 Yacc 能分析指令表语言，在 Yacc 程序中，将指令表的语法按照由上下文无关语法（context-free grammar）描述。也就是说，必须指出一个或者多个语法组（syntactic groupings）以及从语法组的部分构建它的整体的规则。Yacc 要求它的输入必须用巴科斯范式 BNF（Backus Naur Form）来书写和表示。用 BNF 表示的指令表语法都是一种上下文无关语法。

在正式的语言语法规则中，每一种语法单元或组合被称之为符号（symbol）。那些可以通过语法规则被分解成更小结构符号叫做非终结符（nonterminal symbols），不能被再分的符号叫做终结符（terminal symbols）。把同终结符相对应的输入片段叫做记号（token），同单个非终结符相对应的输入片段叫做组（grouping）。

Yacc 程序实际上就是指令表语法规则的说明书，它也是由说明部分、规则部分和子程序部分组成的。

1）说明部分。Yacc 程序的说明部分类似于 Lex 程序的定义部分，可以定义后面动作中使用的类型和变量，以及宏，还可以包含头文件，这些同样包含在%｛和%｝之间，如图 4-5a 所示。在其后可带有 Yacc 声明，每个 Yacc 声明段声明了终结符号和非终结符号的名称。这些终结符可以用上述的 Lex 的返回值。所有这些标记都必须在 Yacc 声明中进行说明。前一部分包含了头文件以及两个函数的前置声明，由于 C 语言要求函数必须在使用之前声明，所以前置声明

yylex 和 yyerror 是必需的。他们都将直接插入到生成的 C 语言中。% token 以及后面的符号声明了终结符，如例子中所示，LINE 表示指令表的标号，AND 和 OUT 表示逻辑与和输出等。终结符必须在此声明才能在后面的 BNF 式中使用。lines 为非终结符，% start 表示以下语法的开始符号为 lines。

2）规则部分。规则部分包括 BNF 格式中的语法规则以及将在识别出相关的语法规则时被执行的 C 代码所写的动作，如图 4-5b 所示。在这里定义的组（lines）是由标号（LINE）、冒号和动作（ACT）组成。“|”表示 lines 可有多个选择。非终结符 ACT 在下面又做了规定，直到所有符号都为终结符为止。每一个语法规则必须用分号结束。lines 后面的语法规则就定义了指令表语言的书写规则和格式。它表示，一行指令表语言必须由标号、冒号或者后面的动作组成，而动作这个非终结符的语法也在下面进行了定义，它可以由命令加操作数组成。如果不是按照这种规定所写的程序，如每行语句中没有标号等，即是错误的语法。

3）子程序部分。如图 4-5c 所示，这一段是 main 主程序，它调用 yyparse 子程序来对输入进行语法分析，而 yyparse 反复地调用 yylex 子程序来获得输入单词，并通过 yyerror 子程序来报告出错信息。

a) 说明部分

```
% {
      #include <stdio.h>
      extern int yylex();
      void yyerror(char * );
% }
%token LINE AND OUT
······
%start lines
```

b) 规则部分

```
% %
lines: LINE“:” ACT “\n”
| TAB“:” “\n”
| error “\n”
{ yyerrok; }
;
ACT:START  VAR
| OUT  VAR
······
;
% %
```

c) 子程序部分

```
void main()
{
  yyparse();
}
 void yyerror (char* msg)
{
  printf (“Error; %s encountered \n”, msg);
}
```

图 4-5 语法分析器示例

a）说明部分 b）规则部分 c）子程序部分

（4）Lex 与 Yacc 的结合 一个由 Yacc 生成的解析器调用 yylex（）函数来获得指令表程序标记。yylex（）可以在 Yacc 中单独编写，也可以由 Lex 来生成，本书将二者结合起来使用。对于由 Lex 生成的词法分析器，和 Yacc 结合使用时，每当 lex（）读取并匹配了一个模式时，就返回一个标记，Yacc 就会获得返回的标记后，进行语法分析。当 Yacc 编译一个带有标记的 .y 文件时，会生成一个头文件，它对每个标记都有 #define 的定义，头文件必须包含在相应的 Lex 源文件中的 C 声明段中。

Lex 与 Yacc 结合生成编译器的步骤：

1）编写一个命名为 mylex. l 的语句表的词法文件，以及一个 myyacc . y 的语法文件（文件名称可以自定义，但文件类型必须为 . l 和 . y）。

2）用 Yacc 运行 myyacc. y 文件，生成 y. tab. c 和 y. tab. h 文件。因为在 Lex 源文件中需要包括 y. tab. h 头文件，利用里面的宏定义，因此要先运行 myyacc. y 文件。执行命令为：$ yacc -d myyacc. y。

3）运行 mylex. l 文件，生产 lex. yy. c 的 C 语言文件。执行命令为：$ lex mylex. l。

4）将 y. tab. c 和 lex. yy. c 连接起来编译，生成可执行文件 compiler。执行命令为：$ cc -o compiler y. tab. c lex. yy. c。

5）运行 compiler，即可对指令表语言进行词法和语法分析，如图 4-6 所示。最后一段为输入错误（包括词法错误和语法错误）时的运行情况。

```
$ ./compiler
000: ST X0.0        001: OT Y0.0  (*output*)    002: xyz
Line:000            Line:002                    Line:002
Colon::             Colon::                     Colon::
Start:ST            Out:OT                      Other symbol:x
Variable:X0.0       Vatiable:Y0.0               Other symbol:y
                                                Other symbol:z
                                                Error is encountered:syntax error
```

图 4-6　语法分析器实例

4.4　Soft-PLC 运行系统设计

计算机的软件系统广泛地应用了模块化的设计原理和方法。将设计系统从结构上分解成许多相互独立的模块，每个模块具有特定的功能属性，分别设计各个部分。然后按一定方式进行组合，构成整体的软件系统。模块化设计通常依据系统所完成的功能来划分模块，各模块的功能应保持相对的独立性。模块化程序结构灵活且适应性强，便于系统的重构、修改和扩充[106]。这种思想应用于 Soft-PLC 的设计符合了开放式系统设计的要求。

本书参照国际标准 IEC61131-3 开发了一套应用于开放式软件数控系统的 Soft-PLC 多任务系统。该系统采用模块化设计，模块结构如图 4-7 所示，其中系统管理模块的任务是在运行每个 PLC 程序之前设定系统参数、建立系统环境，在 PLC 扫描周期内负责调度系统多任务线程，以及监测 PLC 系统运行状况并作出及时响应的系统模块。应用程序执行模块的任务是依次读取程序存储区内的指令，并调用相应的执行函数完成用户所要求的逻辑控制、运动控制功能。输入输出接口模块的任务是按特定的总线协议读写总线上标准 I/O 模块的接口状态数据，并与系统内部全局 I/O 数据存储区进行数据通信的交互。通信模块提

供与其他系统（第三方设备）如其他 PLC 系统、CNC 等进行数据交换的功能。应用程序、数据存储模块是 Soft-PLC 系统中用于管理用户程序存储区及数据存储区的功能模块；Soft-PLC 运行起始阶段程序、数据存储模块向系统申请程序、数据存储内存；PLC 扫描运行阶段向其他模块任务提供访问程序、数据的接口；在 Soft-PLC 系统结束之前释放申请的系统内存。人机界面功能模块是提供操作者、控制器和机器（过程）三者之间的互动，该模块可以形象地显示控制系统及机器的运行状态，用户根据反馈信息来操作系统的运行步骤。

系统运行一个新的 PLC 程序，对应有一组系统管理模块、应用程序存储模块、数据存储模块以及应用程序执行模块，这些模块集合对应于国际标准 IEC61131-3 PLC 软件模型中的每一个资源。在 Visual C ++ 开发环境下，采用 C ++ 语言面向对象方法将 Soft-PLC 系统各功能模块封装成不同类，提供接口函数来实现模块间的通信。运行用户程序时，系统首先会实例化程序运行所需要的资源模块，建立内部的通信机制。当然，系统会建立全局数据区供不同程序、功能模块、系统资源等相互之间访问。其实 Soft-PLC 系统中仅创建一个 I/O 接口模块和一个通信模块，这两者与各资源中模块的通信就是通过全局数据区变量实现的。

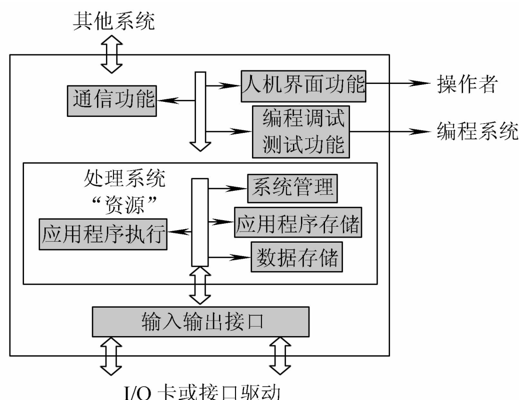


图 4-7　Soft-PLC 系统的基本功能结构模块

按照国际标准 IEC61131-3 PLC 软件模型，资源对应着可以执行用户程序的处理能力，在一个资源内可以有一个或多个任务。对应于多个任务，我们需要为每个任务指定不同的执行时间和优先权。在某一时刻，哪个特定的任务在执行，或者说哪个任务拥有对 CPU 的控制权，取决于任务调度机制、各任务的优先权、执行时间以及系统中当前时刻其他任务的就绪和等待时间。不同的多任务系统划分及调度机制对系统行为的影响是巨大的，表现为即使硬件资源相同也会出现多任务特性的巨大差异。

4.4.1　系统任务划分

Soft-PLC 多任务控制系统中，任务就是完成各个功能的程序模块。各程序模块的具体实现是在功能类实例化对象中初始化创建的线程函数，或者是功能类实例化对象中提供的方法函数。

PLC 系统采用循环扫描工作方式完成用户设定的逻辑控制、运动控制及过程控制等。每个扫描周期内最基本的 PLC 系统需要完成几个任务，包括：PLC 内部工作单元的调度、监控；PLC 与外部设备间的通信；用户程序所完成的工作。各个任务的基本工作状态如图 4-8 所示（如无特别说明，下文中各方块所表达的意义与此处相同）。

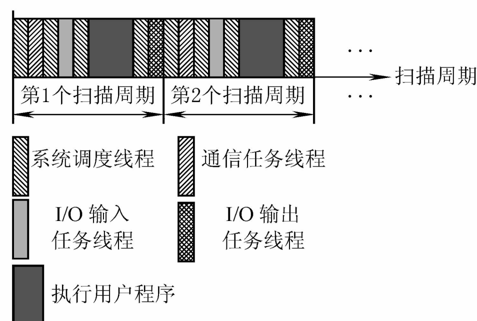


图 4-8　Soft-PLC 基本工作状态

通信功能对象创建一个负责与外部设备以指定格式会话的任务；I/O 接口功能模块定义一个读 I/O 状态到输入数据寄存器，以及将输出数据寄存器中的 I/O 状态写到输出口的任务；系统管理功能相应有个负责调度其他任务执行的任务；应用程序执行功能对应一个从指令内存中首指令地址开始，顺序执行各个指令对应的指令函数的任务。

另外，从用户程序的角度出发，Soft-PLC 提供给用户两个不同等级的任务：基于循环扫描执行程序的任务；基于时间周期执行程序的任务。用户可以将要实现的逻辑控制、运动控制及过程控制程序段按照不同的实时要求，分别交给两个具有不同任务等级的任务完成。

任务按照是否循环执行可分为周期和非周期任务。非周期任务包括程序指令集存储任务和数据存储任务，因为在不考虑 Soft-PLC 在线修改的情况下，这些任务只是在运行某个用户程序的起始阶段完成将用户程序、系统数据写入内存的工作，以及结束阶段完成对内存的释放工作。周期任务就是指在每个循环周期内都被调用的任务，包括通信任务、人机通信任务、输入采集任务、执行用户程序任务、输出任务以及系统调度任务。其中执行用户程序任务又分两个不同等级的任务，即基于循环扫描和基于时间周期（定时执行）的任务。整个 Soft-PLC 系统任务划分如图 4-9 所示。Soft-PLC 最基本的多任务执行流程如图 4-10 所示，这里没有考虑基于时间周期任务

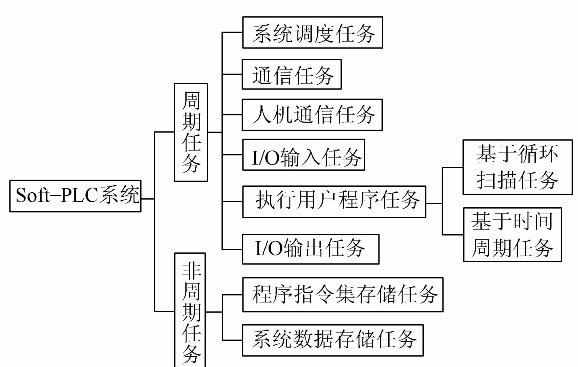


图 4-9　Soft-PLC 系统多任务划分模型

的执行。因为该任务只是在 PLC 周期扫描阶段，定时地被唤醒并由操作系统中断基于循环扫描任务，继而执行自己内部的操作后进入等待下一个周期的状态，基于循环扫描任务重新占据 CPU 资源并继续自己的循环流程。

4.4.2 Soft-PLC 基本扫描任务

传统 PLC 只能按自诊断、通信、I/O 读取、用户程序执行、I/O 输出这一固定顺序按顺序扫描、执行程序，对某一程序段不能按用户的实际要求定时执行，这样一来，极大地限制了 PLC 功能的灵活性及其应用范围，不能更好地满足用户的需求。但是，可编程序控制器国际标准 IEC61131-3 允许程序的不同部分在不同的时间、以不同的比率并行执行，极大地扩大了 PLC 的应用范围。本书 Soft-PLC 系统在通用循环扫描执行用户程序任务的基础上，提供一个具有定时扫描执行能力的任务。这个定时扫描执行任务有较高的优先级，主要完成用户所要求的具有强实时性的事件，如可以将机床急停、限位、循环起动及循环暂停逻辑程序安排在这个任务等级中。当然，可以在此 Soft-PLC 功能基础上增加多个具有不同优先级的定时扫描执行的任务。基于循环扫描任务主要执行一些相对实时要求低的控制。但是两个不同任务等级的任务执行用户程序的流程却是相同的。各个任务依次读取对应程序存储区内的指令，然后解释并调用相应的执行函数。两个用户级任务的工作时序如图 4-11 所示。图中最小扫描周期是由用户依据其程序量的大小设定的。

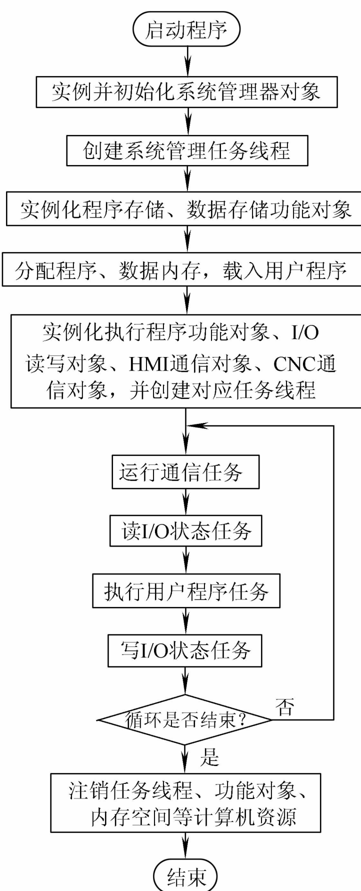


图 4-10 Soft-PLC 系统多任务工作流程

另外，如有需要，可再增加一个基于中断的用户级任务，其优先级最高，工作时序如图 4-12 所示。

4.4.3 任务调度

Soft-PLC 应用软件的结构采用多作业、多任务的组织形式。软件运行的过程中，对任务的调度和监测是必不可少的环节。任务是高内聚低耦合的运行实体，

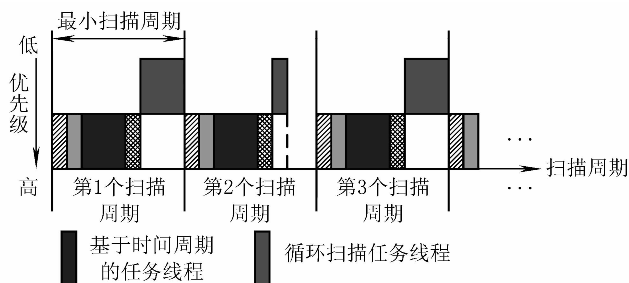


图 4-11　两类用户级任务下的 Soft-PLC 运行时序

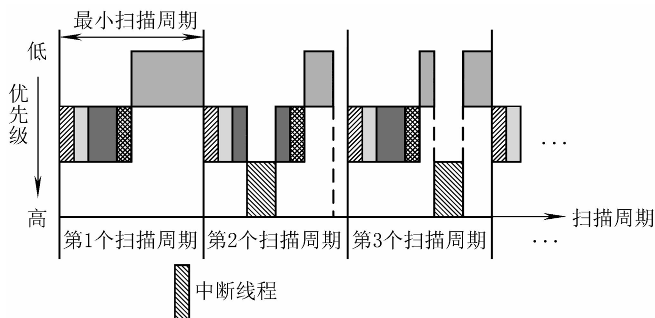


图 4-12　中断任务下的 Soft-PLC 运行时序

是系统中唯一能动的对象，由它们完成具体的应用功能。随着运行工况的不同，各自又会处于就绪、运行、挂起与睡眠四种不同的状态[107]。

控制器系统中任务的调度就是如何平滑地处理多任务之间的状态切换，使它们有条不紊地运行，提高系统整体性能的可靠性。促使任务状态转换的方式从逻辑上可分两种：其一为直接方式，该方式是常规的按优调度的原则，即从完成 PLC 控制器系统功能的角度出发，对每个任务线程按其运行的紧迫程度分配一个优先级。其二为间接方式，该方式通过任务之间的相互关联与相互制约而进行的状态转换。这种任务之间的协调主要体现在：多任务争夺系统中的临界资源而引起的互斥；由于完成相关功能而协调工作带来的同步与通信。间接调度实现了就绪及运行之外的其他状态的切换。RTX 实时子系统中提供事件、互斥、信号量以及共享内存等同步对象来协调工作。

Soft-PLC 系统本来就是循环扫描执行系列任务的控制器系统，在每个循环扫描周期内任务线程执行是有序的。系统为每个任务线程创建一个事件对象来协调多任务线程的执行。只要事件不被触发，线程永远处在睡眠状态。任务线程中使用 RtWaitForSingleObject（）或 RtWaitForMultiObjects（）原语等待同步对象的触发。Soft-PLC 系统管理任务线程依据当前程序执行状态字信息，决定下一个或多个并行运行任务，触发相应的事件对象唤醒睡眠中的线程。例如：当读 I/O

状态任务线程运行结束后，PLC 系统管理任务线程获得 CPU 资源，它读取运行状态字 RunState 值，判断得知须运行程序执行线程并触发 hEvent 对象，唤醒程序执行线程后放弃 CPU 资源，操作系统自动调度程序执行线程。当定时扫描执行的任务线程运行结束，系统管理线程同样会依据状态字确定其后调度写 I/O 状态任务线程。另外 Soft-PLC 运行系统与 HMI 和 CNC 之间则是因为双方对临界资源的访问而采用间接调度方式。

4.4.4 Soft-PLC 数据结构

（1）程序指令集 PLC 程序是实现某种控制的许多指令的集合。PLC 系统依次读取内存中的指令并执行相应的操作，每个指令在 PLC 中都有其对应的执行函数。Soft-PLC 系统用户程序存储功能提供了两种指令集的存储方法，一种采用链表结构存储指令，适合小容量存储，另一种是开辟一片能够容纳 PLC 程序的内存空间，然后从开发系统或 PLC 程序文件中读取指令并依次存放，不同用户级任务 PLC 程序段采取分块存储的措施，各任务可配置子程序内容。所有这些指令在内存中都表示成如下的数据结构形式：

```
typedef struct_INSTRUCTION_ {
    int FuncIndex;                        //指令索引号
    char Param [PARAMS] [LENGTH];         //指令参数
    } Instruction, * pInstruction;
```

结构体中 FuncIndex 是一个整型值，PLC 执行程序是依据这个在 PLC 中唯一标识的索引号调用对应的执行函数，然后将字符型指令参数作为被调函数的实参传递给被调函数。例如：指令“INBIT LI0.7”在内存中的结构数据 FuncIndex 等于 1，字符串 Param［0］内容是“LI0.7”。

Soft-PLC 系统用户程序存储功能提供了两种指令集的存储方法：开辟一片能够容纳 PLC 程序的内存空间，然后从开发系统或 PLC 程序文件中读取指令并依次存放，具体的做法是 vAddress = RtAllocateLockedMemory（size），这种方式适合存储量大的程序，而且执行程序任务线程可以利用定格式指针的加减运算迅速查找指令；另一种方法是采用链表结构存储指令，上面定义的指令结构体和一个节点指针作为链表结构的节点内容，相比较前者它比较适合小容量存储，指令的查找需要通过链表的游历算法来实现，消耗时间比较多，但是链表结构存储指令便于在线修改。

（2）逻辑堆栈 逻辑堆栈是内存中开辟的链表形式的栈数据结构，栈中数据类型是布尔型的。它用来保存指令、功能块逻辑运算后的结果，同时指令、功能块也从逻辑堆栈中获取中间结果参与运算。Soft-PLC 为基于循环扫描的程序执行任务、基于定时周期的程序执行任务各创建一个逻辑堆栈。这样不同的用

户级任务各自管理自己的逻辑数据，避免因相互间中断而保护逻辑堆栈内容。但是在任何程序执行任务内部会定义另外几个逻辑堆栈，用于调度子程序前保护主程序逻辑结果。逻辑堆栈对象是实现系统内部程序、功能及功能块之间通信的局部变量。各用户级任务间的切换、中断及数据的保存由操作系统自己完成。

PLC 系统提供的指令系统中也包含对逻辑堆栈执行操作的指令。例如：执行 Y = X0 | （X1&X2）逻辑关系，对应指令顺序就是 INBIT X0；INBIT X1；ANDBIT X2；ORFIR2。第 1 步读 X0 状态并存入逻辑堆栈；第 2 步读 X1 状态并压入堆栈；第 3 步读 X2 状态并与逻辑堆栈栈顶元素“与”操作；第 4 步是将栈顶两个元素“或”操作。最后一步就是完全对逻辑堆栈的指令操作。依据逻辑堆栈的这些特点设计一个对象模型类如下，逻辑堆栈类中提供对逻辑堆栈操作的方法。

```
class CLogicStack    {
    public：bool Andfirst2（）;
            bool Orfirst2（）；…                //与逻辑堆栈相关的系统指令
            ListNode<STACKTYPE> * firstPtr;
            ListNode<STACKTYPE> * lastPtr；… //栈区地址属性
            …        };
```

（3）I/O 寄存器　I/O 寄存器也称为内存 I/O 数据区，用于存储外部开关量状态、模拟量数值，以及内部继电器、定时器、计数器 I/O 状态。这类数据信息作为系统的全局变量或直接变量，可以供 Soft-PLC 系统中任何任务、程序或功能块访问，由系统数据存储功能模块统一管理。用户级任务、I/O 读写及通信任务均由 I/O 数据读写封装类提供的接口实现。

4.5　人机控制

人机控制沟通控制内核和用户或者其他形式的控制系统使用者，组织和准备控制系统的核心任务，监视执行情况，中断或者执行其他控制，系统操作的自动执行，系统部件的诊断、检测、仿真、维护，以及提供相关信息等[108,109]。

4.5.1　HMI 与 Soft-PLC 的通信

人机界面（Human Machine Interface，HMI）是提供给用户执行操作控制的界面平台，与控制系统实时进程之间传递着实时或非实时的信息。

HMI 是运行在 Win32 系统环境下的界面应用程序，而 Soft-PLC 是运行在实

时子系统 RTX 环境下的应用程序。两个不同的环境拥有各自的控制内核，实时环境下运行的线程优先级要高于所有 Win32 环境中的线程。实时子系统 RTX 提供 IPC（进程间通信）对象以及一些 RTAPI 函数，为控制系统开发人员实现进程间、不同环境中进程间的通信。包括事件、信号量、互斥、共享内存等 IPC 对象，以及创建对象、触发对象、注销对象等函数。HMI 进程与 Soft-PLC 系统之间的通信是在建立共享内存的基础上，通过互斥协调两者之间的通信。这也是前面分析的多任务调度中间接调度方式的一种，属于因对临界资源访问引起的互斥。互斥对象协调 HMI 与 Soft-PLC 通过共享内存通信，如图 4-13 所示。控制系统具体实现过程中发现，HMI 与 Soft-PLC 运行系统之间的通信并不是直接应用 RTX 提供的函数和对象那么简单。

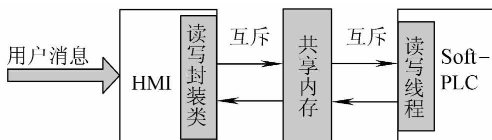


图 4-13　HMI 与 Soft-PLC 通信

首先要考虑实时响应用户的输入，例如：用户通过鼠标按键操作某个界面按钮，那么就得保证鼠标单击事件首先在界面程序中得到响应。这点与实时子系统进程优先级高于 Win32 进程优先级相矛盾，PLC 系统是一个循环扫描多任务执行的控制系统，启动 PLC 运行系统后，RTX 子系统任务调度内核中就会时刻充实着实时线程，那么 Win32 系统几乎得不到 CPU 资源，也就无从执行其环境下的应用程序，用户通过鼠标单击事件得不到响应。所以在单 CPU、单操作系统的支撑平台上，首先得解决 Win32 与 RTX 环境使用 CPU 资源的分配问题。

其次是 HMI 对共享内存访问的方法，HMI 与 Soft-PLC 运行系统两者之间的信息交换有事件、数据等，对于数据的交互按事先设定的格式读写。Soft-PLC 运行系统中有个专门负责读写共享内存的任务线程，将共享内存区的数据传递到系统内部指定的 I/O 数据存储区。这类似 I/O 接口模块任务封装了底层的 I/O 接口信息，系统外部的更改并不会太多地影响系统改变。HMI 应用程序的设计中，采用了用户界面与系统数据相分离的原则，使用户界面完全独立成为一个子系统，从而实现了系统操作界面的用户定制。用户界面操作涉及的数据大部分封装在一个共享内存数据类中，通过类提供的数据读写接口进行访问。

本书在解决首要问题的过程中提出了三种解决方案：

（1）给定 PLC 系统最小循环扫描周期　给定最小循环扫描周期具体的周期时间是由用户依据其程序量的大小设定的，最小扫描时间给定原则是保证绝大多数扫描周期小于最小设定时间。采用最小循环扫描时间设定方法，PLC 运行时出现的工作时序状况如图 4-14 所示。

其中第 1 个周期内 PLC 正常扫描执行，用时小于最小设定周期，剩余的时

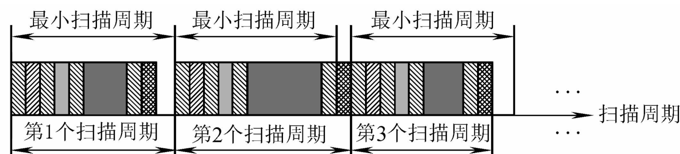


图 4-14 给定最小扫描周期情况下 PLC 运行时序

间内可以用于响应 Win32 应用程序；第 2 个周期内 PLC 程序执行了子程序，扫描时间消耗大于最小设定周期，没有时间空缺，但 Soft-PLC 系统将继续执行下一个周期；第 2 周期结束后立即执行第 3 周期的同时，重新启动最小循环扫描时间定时。结合实验以及方法间的比较分析，这种方法适用于用户程序包含子程序段少以及子程序段多但调用不频繁的情况。

(2) 定时挂起 PLC 所有多任务线程 Soft-PLC 系统设定一个定时触发的任务，任务负责挂起 Soft-PLC 系统中所有的任务线程，无论其是否处在运行（被此任务中断）、就绪以及等待状态。其后系统等待少许时间，RTX 调度内核中无线程就绪的情况下，主动将 CPU 资源让于 Win32 环境，响应用户输入。PLC 系统循环工作时序如图 4-15 所示。这种方法需要考虑定时周期和挂起中断时间片对 Soft-PLC 扫描周期的影响，频繁的定时中断和长时间挂起会增加每个扫描周期的时间。系统可以提供参数供用户权衡设定这两个时间片的大小。

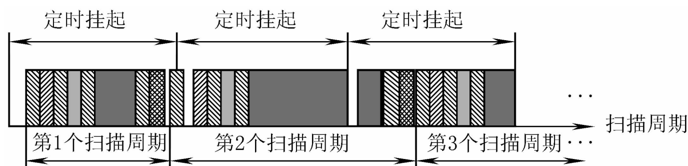


图 4-15 定时挂起情况下 Soft-PLC 运行时序

(3) 利用 MUTEX（互斥对象）特性 互斥对象允许创建它的线程可以立即获得其控制权，在系统中获得互斥对象控制权是取得对共享内存临界资源访问的前提条件。HMI 应用程序利用 RTAPI 方法函数 RtCreateMutex（）创建 MUTEX 对象，在 Soft-PLC 系统中打开此特定名称标识的互斥对象句柄。对临界区共享内存的访问是通过 RtWaitForSingleObject（）原语等待互斥对象的控制权，该原语被用在 Soft-PLC 与 HMI 通信的线程，以及 HMI 内的实现对共享内存读写的封装类接口函数中。因为 RTX 实时系统采用优先级提升的方法解决优先级倒置问题，当一个较高优先级的线程等待一个较低优先级正在使用的资源，则这个较低优先级线程的有效优先级被提升到那个较高优先级线程的水平，所以通过互斥资源协调 HMI 和 PLC 间的通信，实时 PLC 读共享内存线程需等待 HMI 读写线程，HMI 的用户响应不会被中断。但是频繁地响应界面点击输入会占据 CPU 资源，导致 PLC 系统实时运行扫描时间的增加。

HMI 与 PLC 在这种方式下通信，必须将共享内存和互斥对象的创建工作留在 HMI 应用软件内，否则人机界面输入输出几乎得不到响应。项目中最终采用这种方法实现 HMI 与 Soft-PLC 的通信。

4.5.2 HMI 设计

界面是用户和系统交流最直接的平台，用户界面的相关事件和消息的处理可以通过系统内部的各功能接口函数实现。与 Soft-PLC 系统进行数据信息量交互的人机界面，包含面板控制量输入、各种状态显示及实时系统的控制。其中开关量输入如手动、自动、启动、急停、主轴正转、主轴停、主轴反转、进给循环、进给暂停等，其状态是作为 PLC 逻辑控制程序的输入量。显示灯如手动状态、自动状态、回原点、X 原点、Y 原点、Z 原点等，是 PLC 控制程序的输出量。控制信号按钮如 PLC 启动、停止等是控制 PLC 软件运行和关闭。制表页内有用于显示 PLC 系统运行状态的编辑框，有用于设置 CNC 及 PLC 系统参数的对话框。其实 Visual C + + 开发环境提供很多开发界面的函数和工具，如基层的 WinAPI 函数和高级的封装 MFC，开发人员可以随心所欲地实现自己的界面。

但是在实现和 Soft-PLC 实时控制系统通信的 HMI 界面内，还要考虑如何使得各个界面元素与通信数据联系起来。若将共享内存看成是数据库，那么界面元素内部需要实现对数据库系统数据的操作。例如：界面上手动按钮，按钮首先要接受鼠标单击消息，Windows 系统响应消息机制调用按钮的鼠标单击消息处理函数，同时作为 PLC 输入量的控制 NC 运行方式的 I/O 状态，在共享内存对应位置的值也应该更改，PLC 系统才会通过读写线程读出该状态的改变。

本书设计的人机界面见第 6 章。

4.6 小结

本章首先参考国际标准 IEC61131-3，运用 Lex 和 Yacc 编译软件，开发了一套 Soft-PLC 编程开发系统，包括编辑器、解释、编译模块等。根据编译的原理和技术，将编译模块分为四个部分：词法分析，语法分析，语义分析和生成目标代码。将所开发的指令表的词法分析程序和语法分析程序加入到 Soft-PLC 开发系统的编译模块中，实现了对指令表编程语言的扫描和语法分析。

接着给出了一种基于 RTX 的 Soft-PLC 多任务运行系统的实现方法，该方法采用面向对象技术及模块化的思想构造而成。由于采用 RTX 作为操作系统的实时扩展，整个系统的实时性得到保障。解决了在单 CPU 系统资源情况下，Win32 环境中 HMI 进程与 RTX 实时环境中 Soft-PLC 并行执行的问题，提出设定最小扫描时间、定时挂起以及基于 MUTEX 特性的三种解决方法。利用指定内容格式的

共享内存实现了 Soft-PLC 与 HMI、CNC 之间快速通信。

在第 6 章利用开发系统编制基于开放式五轴数控铣床的基本逻辑控制梯形图，转换为目标代码传递到机床的 Soft-PLC 运行系统中，实现了控制功能，证明了开发系统生成目标代码的正确性和可行性以及运行系统的实时性和可靠性。

第 5 章　Open-CNC 制造系统多轴精密插值机理

5.1　概述

在数控机床上进行曲面的实际加工时，需由数控系统对数控程序（NC 代码）进行译码、插值后将获得的数据传送给各轴的驱动器，再经过位置控制以及速度控制等环节后驱动电动机完成运动。其中插值、位置控制和速度控制环节必须实时完成。在数控系统诸多实时任务中，实时插值计算是最重要、最核心的工作。通过插值，数控系统可以控制刀具相对于工件以给定速度、沿给定的路径运动以完成工件轮廓的加工。插值任务由插值器完成。对硬件数控而言，插值器由硬件构成，而对 Open-CNC 系统而言，插值器完全由计算机软件程序实现。CNC 系统插值程序的运行时间和计算精度影响整个系统的性能指标，因此，需要寻求简单而有效的插值算法。

本书的插值方法是基于数据采样插值算法，由软件实现。该方法由粗插值和精插值两个步骤组成。在粗插值阶段，采用时间分割思想，根据 NC 代码给定的进给速度 F、数控系统的插值周期 T 及人机界面（HMI）给定的进给倍率 φ，将工件的轮廓曲线分割成一段段的轮廓步长 L_T（也称为插值步长），$L_T = \varphi FT$，然后根据插值步长，计算出每个插值周期各轴的坐标增量，进而计算出插值点的位置坐标。在精插值阶段，要根据位置反馈采样周期的大小，由伺服系统完成。本书的精插值工作、位置控制以及速度控制环节均由系统装备的 Rexroth IndraDrive 等驱动器完成。

译码后得到的数据如何传送给插值器，是 Open-CNC 系统中实现插值功能的一个前提条件，因此，本章首先论述了译码后的数据流处理方法。线性插值功能在三轴、五轴联动数控系统中都是最常见的，也是必需的，本章对采用梯形加减速的线性插值软件化实现方法进行了深入研究。在开放式数控系统中添加新的插值器是比较容易的，而 NURBS 插值相对于常见的线性插值具有很多优点，为使五轴联动数控系统具备样条曲线直接输出功能，本章实现了一种基于刀心样条曲线和方位样条曲线的五轴双 NURBS 插值器。

5.2　数据流处理

系统的数据流（data stream）是由任务协调器调度流动的。译码器将 NC 代码处理完毕后，以数据链表的形式将数据放入共享内存中，任务协调器适时地将数据取出并以定义的执行步作为每一个节点存入新的数据链表中，插值器运行时便取出其中的数据进行运算处理。

系统为数据传输定义的共享内存结构如下：

```
typedef struct sharedMemory_struct {
      singleStep2    singleStep;
      bool           endFlag;
} shm_struct;
```

译码器在传送 NC 代码时定义一个共享内存指针指向该内存块：shm_struct * pShmSingleStep。

任务协调器是一个实时进程，为实现数据流的可靠传输，需要使用 RTX 的进程间通信方法。RTX 进程（线程）间通信原理与 Windows 相似，需要使用事件（Event）及信号量（Semaphore）等对象来实现进程（线程）同步，区别是 RTX 的实时性强。事件（Event）是 RTX 为线程同步提供的核心对象，可以在进程之间共享。一个事件可以是有信号状态或无信号状态，可以用 RtSetEvent 函数（原语）设置适当的事件为有信号状态。RtWaitForSingleObject 函数可以挂起线程，直到特定的对象变成有信号状态。

传送端线程在启动后就一直处于挂起状态，直到接收端通知它自己已准备好，此时，传送端也会给接收端一个信号，表明自己准备传送数据了。数据是一个节点接着一个节点传送的，节点间的传送以信号量作为标识。简化的数据传送过程如图 5-1 所示。

接收端定义的数据链表为 CListNode <CExecutionStep * > g_programList；其中，CListNode 为一个自定义链表模板类，用于对数据链的处理；CExecutionStep 为一个执行步类，用于定义执行步中要用到的相关数据信息（定义数据结构及指针，分配及销毁相关的内存空间）、函数功能及有限状态机信息等。

执行步类及其子类的指针定义为 CExecutionStep * ms1；CMSExecutionStep * execstep；接收数据时，对每一个 NC 代码段都要先进行插值类型匹配，按类型进行数据处理，然后作为一个节点存入接收端 g_programList 数据链表中，每一个节点对应一个 NC 代码段。其中包含插值类型、代码段起始点坐标、进给模式、进给速度、主轴转速等数据信息。执行步对象的转化过程简述如图 5-2 所示。

```
if( RtWaitForSingleObject(hReceiveStart，INFINITE)
                   ! = WAIT_FAILED) {
    RtSetEvent(hSendStart);
                   // 通知接收端，准备数据传送
   for(int i= 0; i< singleStep_form2.size(); ){
      if(RtWaitForSingleObject(hSemaphoreWrite，
                   INFINITE) ! =WAIT_FAILED){
             (*pShmSingleStep).singleStep =
             singleStep_form2[i];
             (*pShmSingleStep).endFlag = false;
             i++;
             if(i==singleStep_form2.size())
             { (*pShmSingleStep).endFlag = true; }
      }
   }
}
```

图 5-1 简化的数据传送

```
void CReceiver:: convertExecutionStep( ) {
 switch (executionStep.singleStep_type){
   case LINE:
     execstep=(CMSExecutionStep*)
       CreateClassInstance(CMSExecutionStepClass);
     beginPt=new SixPose;
     beginPt->trans.x=executionStep.line_struct
                  .begin_x;
          ...
     g_programList.add_node(ms1);
     break;
    case ARC :
          ...
       ...
  }
}
```

图 5-2 执行步对象的转化

接收数据时，接收端首先通知传送端，接收端已准备好，然后等待传送端发出 hSendStart 信息，收到后便开始接收数据。图 5-3 是简化的数据接收过程。

```
RtSetEvent(hReceiveStart);          //首先通知传送端，接收端已准备好
if(RtWaitForSingleObject(hSendStart，INFINITE) ! =WAIT_FAILED) //等候传送端发出信息
    for(; ; )
   {
      if(RtWaitForSingleObject(hSemaphoreRead，INFINITE) ! =WAIT_FAILED)  {
          executionStep = (*g_pShmSingleStep). singleStep;
          convertExecutionStep();     }
      if((*g_pShmSingleStep). endFlag)  {
          RtReleaseSemaphore(hSemaphoreWrite，1，NULL);
          break; }
    }
```

图 5-3 简化的数据接收

5.3 线性插值机理

现行数控系统，不管是三轴还是五轴联动，采用的最基本的指令都是线性插值。因此，NC 代码中必然会有线性程序段。一方面，由于数据采样插值方法必须考虑如何保证各轴连续运动且同时到达程序段终点的问题。另一方面，为避免各程序段间速度变化给加工表面带来的影响，通常数控系统会在每个程序段内进行加减速处理。对于前者，必须保证在每个程序段各插值轴的运行时间恰好是插值周期的整数倍，这可以通过调整各轴的命令进给率即每个插值周期的进给量来实现。对后者而言由于速度曲线可以多种多样，情况也比较复杂，本节主要考虑梯形加减速的情形。

便于论述，先介绍几个概念。

命令速度 V_{com}，指的是当前 NC 代码段中由 F 参数给定的进给速度值 F 与 HMI 上用户给定的倍率 φ 的乘积值，即 $V_{com}=\varphi F$。另外，在每个插值周期临时保存的当前命令速度记为 V_{com1}。若无特别说明，本章所述的值均为标量值。

当前速度 V_{cur}，指的是在数控系统实时插值当前 NC 代码段的过程中，系统保存的上一个插值周期的实际运行速度；如果插值的是数控程序的第一段代码，且是第一次插值，则该值为 0；若不是第一段代码，则第一次插值时，该值为上一段 NC 代码插值结束后的实际速度值。另外，在每个插值周期临时保存的当前运行速度记为 V_{cur1}。

终点速度 V_{end}，指的是当前 NC 代码段插值结束后应该达到的目标速度值，如果插值的该段 NC 代码不是数控程序的最后一段，则该值等于其命令速度 V_{com}，否则，其值为 0。

目标速度 V_{new}，指的是当前插值周期算出的系统目标速度，其值与下一个插值周期的 V_{cur} 相等。

终点坐标 P_{end}（x_{end}，y_{end}，z_{end}），也即当前 NC 代码段的运动目标点。前文曾提及，插值时，所有的坐标点均已调整为机床坐标系（MCS），下同。

起点坐标 P_{sta}（x_{sta}，y_{sta}，z_{sta}），为当前 NC 代码段的起点，也即上一段的终点。

当前点坐标 P_{cur}（x_{cur}，y_{cur}，z_{cur}），指系统（刀具中心点）此刻在机床坐标系下的坐标位置。

本周期插值目标点 P_{aim}（x_{aim}，y_{aim}，z_{aim}），指的是插值器在当前插值周期算出的系统应该达到的运动目标点。理论上讲，当前插值周期的时间段一结束，系统就应该处于 P_{aim} 的位置。

当前 NC 代码段（程序段）总长度 L_{all}，指的是当前 NC 代码段的起点坐标

与终点坐标之间的空间距离，此值在五轴联动系统中计算旋转轴角度（B，C）时有用。即

$$\begin{aligned} L_{all} &= \left\| P_{sta} - P_{end} \right\| \\ &= \sqrt{(x_{sta}-x_{end})^2 + (y_{sta}-y_{end})^2 + (z_{sta}-z_{end})^2} \end{aligned} \quad (5\text{-}1)$$

程序段余下长度 L_{lft}，指的是当前 NC 代码段在经过若干个插值周期后（可以为 0），当前点坐标与目标点坐标之间的空间距离。即

$$\begin{aligned} L_{lft} &= \left\| P_{cur} - P_{end} \right\| \\ &= \sqrt{(x_{cur}-x_{end})^2 + (y_{cur}-y_{end})^2 + (z_{cur}-z_{end})^2} \end{aligned} \quad (5\text{-}2)$$

另外，系统的加速度值用 a 表示，插值周期为 T，插值步长为 L_T。

从当前速度 V_{cur} 插值到终点速度 V_{end} 预计所需插值周期的个数 N_T 为

$$N_T = [\,|(V_{cur} - V_{end})/(aT)|\,] \quad (5\text{-}3)$$

式中,算子[ ]表示对浮点数取整。

从当前速度 V_{cur} 插值到终点速度 V_{end} 预计所需运行的位移量 L_{est} 为

$$L_{est} = (V_{cur} + V_{end}) N_T T/2 \quad (5\text{-}4)$$

本章的加减速状态表示方法如下：S_{acc} 表示加速状态，S_{eve} 表示匀速状态，S_{dst} 表示减速（到停止）状态，S_{drn} 表示减速到某一非零速度状态，S_{end} 表示本段 NC 代码运行完毕。

5.3.1 制造系统插值过程设计

在系统完成译码并将数据传送给任务协调器后，任务协调器即可调用轴组模块进行插值。

系统根据 HMI 的“启动/CYCLE”命令进入插值状态，首先判断当前 NC 代码段的插值类型，任务协调器根据插值类型将该段代码送到相应的插值器中进行插值。为了保证段间加减速时保持平稳运行，首次插值一段 NC 代码时，需比较 V_{com1} 与 V_{cur1} 的差值 e，若 e 小于给定的值如 0.0001，则说明上一段 NC 代码插值结束后，系统并没有达到给定的命令值 V_{com}，实际达到的值为 V_{cur1}。所以插值本段 NC 代码时的初始值 V_{cur} 应更新为 V_{cur1}。接下来应计算程序段余下长度 L_{lft}，然后进行加减速处理（具体处理过程将在下面论述）。

加减速处理后系统返回插值步长 L_T 值，系统保存好当前的 V_{com1} 与 V_{cur1}，以便插值下一段 NC 代码时调用。

为了算出本周期插值目标点 P_{aim}（x_{aim}，y_{aim}，z_{aim}），还需要算出 P_{cur} 与 P_{com} 构成空间直线的方向余弦值。设 l_x，l_y，l_z 为其方向余弦，则

$$l_x = (x_{end} - x_{cur})/L_{lft},\ l_y = (y_{end} - y_{cur})/L_{lft},$$
$$l_z = (z_{end} - z_{cur})/L_{lft} \tag{5-5}$$

由此插值目标点 P_{aim} 在机床坐标系下的空间坐标为

$$x_{aim} = x_{cur} + l_x L_T,$$
$$y_{aim} = y_{cur} + l_y L_T,\ z_{aim} = z_{cur} + l_z L_T \tag{5-6}$$

具体的插值过程如图 5-4 所示。

对于五轴联动系统，还需算出该插值点处的旋转轴角度（B，C）值。当前 NC 代码段（B，C）的起点记为（b_{sta}，c_{sta}），当前点记为（b_{cur}，c_{cur}），终点记为（b_{end}，c_{end}），当前插值周期的目标点记为（b_{aim}，c_{aim}），当前 NC 代码段给定的起点坐标与终点坐标的差值为

$$B_{all} = \| B_{sta} - B_{end} \|,$$
$$C_{all} = \| C_{sta} - C_{end} \| \tag{5-7}$$

则当前插值周期下（B，C）插值目标点在机床坐标系下的空间坐标为

$$b_{aim} = b_{cur} + B_{all} L_T / L_{all},$$
$$c_{aim} = c_{cur} + C_{all} L_T / L_{all} \tag{5-8}$$

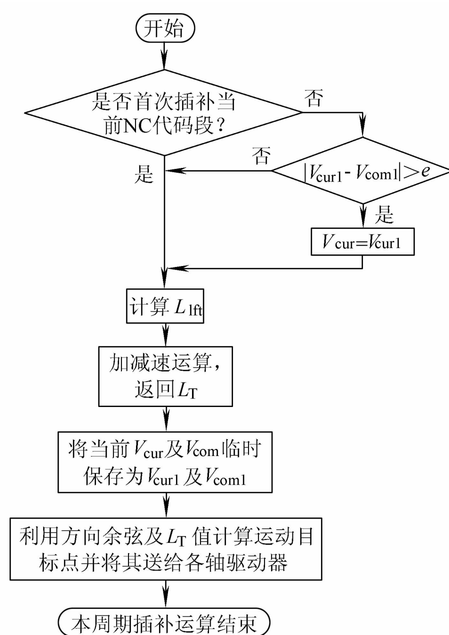


图 5-4　线性程序段的插值过程

5.3.2　进给加减速实现方法

传统的梯形加减速方法需预先确定出加速区、匀速区及减速区等各区段的运动位移量，但是，在实际插值时由于给定的进给倍率，以及 NC 代码段间速度的变化，这些位移量往往不能事先精确给出。本书给出一种适合于 Open-CNC 系统的加减速算法，这种方法的核心思想是在每个插值周期中，系统均实时刷新当前运行速度，并确定当前 NC 代码段中余下的待运行位移量 L_{lft}，就可确定本插值周期的插值步长 L_T。这种方法在软件化数控系统中能极大地简化运算过程，提高系统运行效率。后文进行的加工实验证明了本方法的可行性和正确性。具体的实现方法及过程如下：

首先从数据内存区取出本段 NC 代码的命令速度 V_{com}、当前速度 V_{cur}、终点

速度 V_{end}及加速度值 a，判断当前的加减速状态，若为 S_{end}，表明插值已完成，则令插值步长 $L_T=0$，$V_{new}=0$，然后退出加减速过程。否则，算出 N_T、L_{est}等相关值，接下来进行相关处理。

若 $L_{lft} \leqslant (V_{end}+aT)T$，则说明当前 NC 代码段在本插值周期内即可运行完，置加减速状态标志为 S_{end}，令插值步长 $L_T=L_{lft}$。但是在这种状况下，对速度的处理又分为两种情况：

1）$V_{cur}<V_{end}$，说明目前还处于加速状态下，但是没有加速到终点速度 V_{end}，就已经运行到终点 P_{end}了。则为了使段间加减速保持平稳，应使系统在当前插值周期的目标速度 $V_{new}=V_{cur}$，而不是 $V_{new}=V_{end}$；接下来系统更新 V_{new}，以备下一个 NC 代码段使用。

2）$V_{cur} \geqslant V_{end}$，系统目前处于匀速或减速状态下，应使当前插值周期的目标速度 $V_{new}=V_{end}$。

若 $L_{lft}>(V_{end}+aT)T$，则说明当前 NC 代码段还需经过若干个插值周期才可运行完，但是在这种状况下，对速度的处理又可分为三种情况：

1）$V_{cur} \leqslant V_{com}$，且 $L_{lft} \leqslant L_{est}$，则系统处于减速状态，置 S_{dst}（下一个插值周期判断处于相同运动状态的条件为 $V_{cur} \leqslant V_{com}$，且 S_{dst}），并令 $V_{new}=V_{cur}-aT$；另外还要修正一下，若算得 $V_{new}<V_{end}$，则使 $V_{new}=V_{end}$。接下来计算插值步长：$L_T=V_{new}T$；为了防止“超程”，即运动到目标点以外，还应校验一下，若 $L_T>L_{lft}$，则将 L_{lft}赋值给 L_T，并更新 $V_{new}=L_{lft}/T$，并置 S_{end}。

按以上方法计算插值步长，会遇到一种特殊情况：若插值的是最后一段 NC 代码，即 $V_{end}=0$，按以上方法给 V_{new}赋值时，会出现 $V_{new}=0$ 的情况，则算得的插值步长也为0，会产生“失步”现象，即没有运动到目标点就停止不前了。不过，这种现象发生时，会有 $L_{est}=0$。所以，应当再增加一个判断，当 $L_{lft}>L_{est}$时，令 $V_{new}=V_{cur}$即可。

2）$V_{cur}>V_{com}$，则为减速运行非最后一段数控代码，置 S_{drn}，且令 $V_{new}=V_{cur}-aT$；另外还要修正一下，若算得 $V_{new}<V_{com}$，则使 $V_{new}=V_{com}$。同样，插值步长 $L_T=V_{new}T$。

按以上方法计算插值步长，也可能会遇到一种特殊情况：系统已运行到本段 NC 代码的终点（目标点）P_{end}，但是速度还没有减到目标速度 V_{end}，此时 $L_T \geqslant L_{lft}$。为了在运行下一段 NC 代码时，系统保持平稳状态，需调整 $V_{new}=L_{lft}/T$，并重算 $L_T=L_{lft}$，此时本段 NC 代码运行完毕，置 S_{end}。

3）若以上两种条件不满足，剩下的状态就是加速了。显然，应该使 $V_{new}=V_{cur}+aT$，并置 S_{acc}。当然，还需要判断一下 V_{new}是否已经超过命令速度，若 $V_{new}>V_{com}$，则令 $V_{new}=V_{com}$，并置 S_{eve}。插值步长 $L_T=V_{new}T$。

按以上方法计算插值步长，也可能会到一种特殊情况：系统已运行到本段

NC 代码的终点（目标点）P_{end}，但是速度还没有增加到目标速度 V_{end}，此时 $L_T \geqslant L_{lft}$。为了运行下一段 NC 代码时，系统保持平稳状态，需调整 $V_{new}=L_{lft}/T$，并重算 $L_T=L_{lft}$，此时本段 NC 代码运行完毕，置 S_{end}。

以上各步骤结束后均更新保存 V_{new} 值及加减速状态 S，最后返回 L_T 值。计算过程如图 5-5 所示（图中斜体字为注释的内容）。

图 5-5　梯形加减速处理过程

5.4　多轴样条精密插值的实现机理

目前，复杂曲面零部件的生产和制造在航空航天、汽车、轮船、刀具模具等行业具有特别重要的现实意义。这类零件的高速高精控制加工一般在五轴联动数控机床上完成。另外，在 CAD/CAM 软件中，零件自由曲面的设计常采用非均匀有理 B 样条（NURBS）表示[110,111]。这就要求计算机数控（CNC）系统应该具备五轴联动 NURBS 样条曲线直接实时生成功能。NURBS 技术在 CAD/CAM 领域已经取得了比较成功的应用，而在 CNC 领域的应用却相对滞后，目前只有 FANUC、SIEMENS、三菱等少数高档数控系统支持 NURBS 样条曲线插值[112]。如参考文献［113］中所指出的那样，NURBS 插值相对于常见的线性插值具有很多优点，因此，研究 CNC 系统中的 NURBS 插值方法意义深刻。目前有关 NURBS 样条曲线插值方法的研究大多局限于三坐标联动[114,115]，在有关五轴联动 NURBS 插值技术方面未见有公开发表。国外 CNC 系统商都声称自己已经具有五轴联动 NURBS 插值功能，但对于这些功能却纷纷对华采取禁运，因此，国内进口的五轴联动数控系统大多只有线性插值等简单功能。自行研制开发五轴联动数控系统时，为使系统支持 NURBS 样条插值就必须进行这方面的研究。本书将首先提出一种适合于五轴联动加工的双 NURBS 曲线插值格式，然后将对具有该格式数控代码的生成以及插值过程进行介绍。

5.4.1　样条曲线表示方法

三维空间中的一条参数曲线的一般表达式为

$$\boldsymbol{C}(u)=x(u)\boldsymbol{i}+y(u)\boldsymbol{j}+z(u)\boldsymbol{k}\quad 0\leqslant u\leqslant 1 \tag{5-9}$$

式中，u 为曲线的形参；$\boldsymbol{i}$、$\boldsymbol{j}$、$\boldsymbol{k}$ 分别为 x、y、z 轴向的单位矢量。

一条 p 次 NURBS 曲线可由下式定义[116]：

$$\begin{aligned}\boldsymbol{C}(u)&=[x(u)y(u)z(u)]^{\mathrm{T}}\\&=\frac{\sum_{i=0}^{n}N_{i,p}(u)W_i\boldsymbol{P}_i}{\sum_{i=0}^{n}N_{i,p}(u)W_i}\\&=\sum_{i=0}^{n}\boldsymbol{P}_iR_{i,p}(u)\quad 0\leqslant u\leqslant 1\end{aligned} \tag{5-10}$$

$$R_{i,p}(u)=\frac{N_{i,p}(u)W_i}{\sum_{i=0}^{n}N_{i,p}(u)W_i} \tag{5-11}$$

式中，$\boldsymbol{P}_i$ 为第 i 个三维控制点；w_i 为 $\boldsymbol{P}_i$ 对应的权重；$(n+1)$ 为控制点数；$N_{i,p}(u)$

为 p 次 B 样条基函数，并由如下递推公式定义：

$$\begin{cases} N_{i,0}=\begin{cases}1 & (u_i \leqslant u < u_{i+1}) \\ 0 & (\text{其他})\end{cases} \\ N_{i,p}(u)=\dfrac{u-u_i}{u_{i+p}-u_i}N_{i,p-1}(u)+\dfrac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}}N_{i+1,p-1}(u) \\ \dfrac{0}{0}=0 \quad (\text{规定}) \end{cases} \tag{5-12}$$

式中，$[u_i, \cdots, u_{i+p+1}]$ 为节点矢量。

5.4.2 双 NURBS 曲线插值格式

在数控机床上实现 NURBS 插值有两种途径：

1）在数控系统内部首先将 CAM 得到的线性刀具加工轨迹在一定容差范围内插值（或拟合）成 NURBS 插值刀具轨迹，然后 CNC 系统执行 NURBS 实时插值运算。

2）CAM 系统进行数控编程时将 CAD 中由 NURBS 曲线定义的几何模型直接转化成含有 NURBS 插值代码的 NC 文件，数控系统能够识别这些代码并进行 NURBS 插值运算。

显然，与第 1）种方法相比，第 2）种方法不存在刀轨转化时的拟合误差、精度也更高。本书主要讨论的是第 2）种 NURBS 插值方式。

现行三坐标 NURBS 指令还没有统一的标准格式，比较典型的数控代码插值格式如图 5-6a 所示。

G06.2 P_； X_ Y_ Z_ R_ K_ F_； X_ Y_ Z_ R_ K_； …… K_； K_； ……	G06.2：NURBS插补数控代码 P：曲线次数 XYZ：控制点坐标 R：权值 K：节点值

a)

G06.5 P_: X_ Y_ Z_ U_ V_ W_ R_ K_ F_； X_ Y_ Z_ U_ V_ W_ R_ K_； …… X_ Y_ Z_ U_ V_ W_ R_ K_； K_； K_； ……

b)

图 5-6 NURBS 数控代码插值格式

a）典型的数控代码插值格式 b）五轴双 NURBS 样条曲线插值指令格式

程序段由 G06.2 作为 NURBS 插值开始的标志，数控装置读入其后的数据从而实现 NURBS 插值。然而，这种格式的代码所提供的信息量明显不足以进行五轴联动插值加工。这是因为三轴联动刀具轴向固定，不必考虑刀轴的方位问题，而五轴联动加工中每个插值周期刀具的轴向方位必须实时确定出。这就要求必

须在 NURBS 插值代码中加入可以确定刀轴方位矢量的数据信息。用 CAM 系统进行五轴数控编程阶段，根据所选取的刀具路径生成方法、初始给定的刀倾角和刀转角以及干涉检验修正后，可以分别计算出刀心点坐标和刀轴单位矢量。本书所提出的双 NURBS 曲线插值格式就是在现行使用的三坐标 NURBS 指令格式的基础上进行扩充，加入一个表示方位的 NURBS 样条曲线。加入这个方位样条后，加上原有的刀心点样条，插值格式中就有两个样条。图 5-6b 所示为本书提出的五轴双 NURBS 样条曲线插值指令格式，该格式以 G06.5 作为插值开始的标识。XYZ_表示刀心点样条的控制点坐标；方位样条的控制点坐标用关键字 UVW 表达，其对应权值和定义样条的节点矢量与刀心点样条相同。

5.4.3　加工轨迹的 NURBS 拟合方法

由于通用 CAD/CAM 软件如 Unigraphics 等不能生成含有所提出的五轴联动双 NURBS 样条插值格式的数控加工代码，因此 CAD/CAM 模块需自行开发完成。图 5-7 所示为用自行开发的 CAD/CAM 模块生成数控代码的流程。

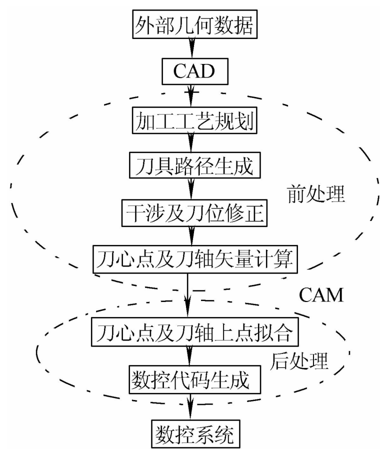


图 5-7　新样条格式代码的生成过程

模块的 CAD 部分提供与其他 CAD/CAM 软件的接口，充分运用这些优秀 CAD/CAM 软件的几何造型功能。模块的 CAM 部分在进行刀具路径规划[117,118]、干涉检验[119]等前处理环节后，计算出刀心点坐标及其相应的刀轴单位矢量，后置处理环节分别将这些刀心点和刀轴单位矢量进行 NURBS 拟合[120]成刀心点样条和方位样条并输出含有双 NURBS 样条曲线插值格式的数控加工代码。

5.4.4　双 NURBS 曲线数据流处理

（1）译码器的实现　自行研制的开放式数控系统的模块化结构如图 2-5 所示。由于系统的开放性，使得在其中集成五轴 NURBS 译码及插值器就显得比较容易，仅需修改任务生成模块及轴组模块即可实现。

任务生成模块中用到的数据结构（用 C++语言描述）如图 5-8 所示。

```
typedef std::deque<singleStep>singleStep_deque;
```

模块中添加 dnurbs 结构用于五轴 NURBS 的译码，当该模块读取到样条程序段，即读到“G06.5”时，则开辟一段存储空间给该程序段，并把段内信息提取

```
typedef enum {LINE，DNURBS} SingleStep_TYPE；//定义运动段类型
typedef struct dnurbs_struct{
    double start_rate, traverse_rate, end_rate；  //运动段的进给速度
    int p；                                        // NURBS曲线的次数
    double r[20]，kv[26]；                         // 保存权重和节点矢量的数组
    double clx[20], cly[20], clz[20]；  //保存刀心控制点坐标的数组
    double tclx[20], tcly[20], tclz[20]；//保存刀轴方位样条控制点坐标的数组
    ...
} dnurbs；
typedef struct singleStep_struct{
    SingleStep_TYPE singleStep_type；
    line line_struct；                             // 保存直线段信息的结构
    dnurbs dnurbs_struct；                         // 保存双NURBS段信息的结构
    ...
} singleStep ；
```

图 5-8　任务生成模块中用到的数据结构

出来。从每个运动段中提取的信息放入 singleStep_deque 队列中，然后被传送到轴组模块中进行实时插值。

（2）插值器的实现　双 NURBS 样条曲线实时插值的任务是要根据当前插值周期插值点对应的参数值 u_k 计算出在下一周期的参数值 u_{k+1}，同时将得到的刀轴矢量转化为工作台的摆转运动坐标 B、C（本书所用机床为工作台摆转式五轴联动数控铣床，如图 5-9 所示）。

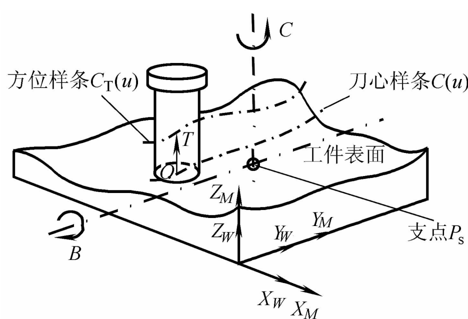


图 5-9　工作台摆转式五轴 CNC 机床

假设某 NURBS 插值程序段的刀心样条和方位样条分别为 $\boldsymbol{C}(u)$ 和 $\boldsymbol{C}_T(u)$，建立在相同的节点矢量上。时间函数 u 为曲线参数，并且 $u(t_k)=u_k$，$u(t_{k+1})=u_{k+1}$。将参数 u 对时间 t 按泰勒公式展开，得到一阶逼近插值算法，其导数公式为[121]

$$u_{k+1}=u_k+\left.\frac{\mathrm{d}u}{\mathrm{d}t}\right|_{t=t_k}(t_{k+1}-t_k)+E(t_{k+1}-t_k)^2 \tag{5-13}$$

式中，$E(t_{k+1}-t_k)^2$ 为一阶逼近误差。

插值周期为 T_s，沿曲线的进给速度定义为

$$V(u)=\frac{\mathrm{d}\boldsymbol{C}(u)}{\mathrm{d}t}=\left(\frac{\mathrm{d}\boldsymbol{C}(u)}{\mathrm{d}u}\right)\left(\frac{\mathrm{d}u}{\mathrm{d}t}\right) \tag{5-14}$$

由于

$$V(u_k)=\left\|\frac{\mathrm{d}\boldsymbol{C}(u)}{\mathrm{d}t}\right\|_{u=u_k} \tag{5-15}$$

所以 u 对 t 的一阶导数可表示为

$$\left.\frac{\mathrm{d}u}{\mathrm{d}t}\right|_{t=t_k}=\frac{V(u_k)}{\left\|\frac{\mathrm{d}\boldsymbol{C}(u)}{\mathrm{d}u}\right\|_{u=u_k}} \tag{5-16}$$

因此将式（5-16）代入式（5-13），忽略高阶项，由式（5-13）可得一阶插值算法

$$u_{k+1}=u_k+\frac{V(u_k)T_{\mathrm{s}}}{\left\|\frac{\mathrm{d}\boldsymbol{C}(u)}{\mathrm{d}u}\right\|_{u=u_k}} \tag{5-17}$$

同样的，二阶插值算法为

$$u_{k+1}=u_k+\frac{V(u_k)T_{\mathrm{s}}}{\left\|\frac{\mathrm{d}\boldsymbol{C}(u)}{\mathrm{d}u}\right\|_{u=u_k}}-\frac{\left\|V^2(u_k)T_{\mathrm{s}}^2\left(\frac{\mathrm{d}\boldsymbol{C}(u)}{\mathrm{d}u}\frac{\mathrm{d}^2\boldsymbol{C}(u)}{\mathrm{d}u^2}\right)\right\|_{u=u_k}}{2\left\|\frac{\mathrm{d}\boldsymbol{C}(u)}{\mathrm{d}u}\right\|_{u=u_k}^4} \tag{5-18}$$

$\boldsymbol{C}(u)$ 对 u 的一、二阶导数分别为

$$\begin{aligned}\frac{\mathrm{d}\boldsymbol{C}(u)}{\mathrm{d}u}&=\sum_{i=0}^{n}\boldsymbol{P}_iR'_{i,p}(u)\\&=\frac{\sum_{i=0}^{n}N'_{i,p}(u)W_i\boldsymbol{P}_i}{\sum_{i=0}^{n}N_{i,p}(u)W_i}-\frac{\sum_{i=0}^{n}N'_{i,p}(u)W_i\sum_{i=0}^{n}N_{i,p}(u)W_i\boldsymbol{P}_i}{\left(\sum_{i=0}^{n}N_{i,p}(u)W_i\right)^2}\end{aligned} \tag{5-19}$$

$$\begin{aligned}\frac{\mathrm{d}^2\boldsymbol{C}(u)}{\mathrm{d}u^2}=&\frac{\sum_{i=1}^{n}N_{i,p}(u)W_i\sum_{i=0}^{n}N''_{i,p}(u)W_i\boldsymbol{P}_i-\sum_{i=0}^{n}N''_{i,p}(u)W_i\sum_{i=1}^{n}N_{i,p}(u)W_i\boldsymbol{P}_i}{\left(\sum_{i=0}^{n}N_{i,p}(u)W_i\right)^2}\\&-\frac{2\sum_{i=0}^{n}N'_{i,p}(u)W_i\left(\sum_{i=0}^{n}N_{i,p}(u)W_i\sum_{i=0}^{n}N'_{i,p}(u)W_i\boldsymbol{P}_i-\sum_{i=0}^{n}N_{i,p}(u)W_i\boldsymbol{P}_i\sum_{i=0}^{n}N'_{i,p}(u)W_i\right)}{\left(\sum_{i=0}^{n}N_{i,p}(u)W_i\right)^3}\end{aligned} \tag{5-20}$$

式中，$N_{i,p}(u)$ 的 m 阶导数通用计算公式为

$$N_{i,p}^{(m)}(u)=p\left[\frac{N_{i,p}^{m-1}(u)}{u_{i+p}-u_i}-\frac{N_{i+1,p}^{m-1}(u)}{u_{i+p+1}-u_{i+1}}\right] \tag{5-21}$$

且

$$\frac{\mathrm{d}\boldsymbol{C}(u)}{\mathrm{d}u}=\sqrt{(x')^2+(y')^2+(z')^2} \tag{5-22}$$

式中，

$$x=C_x(u),\ y=C_y(u),\ z=C_z(u),$$

$$x'=\frac{\mathrm{d}C_x(u)}{\mathrm{d}u},\ y'=\frac{\mathrm{d}C_y(u)}{\mathrm{d}u},\ z'=\frac{\mathrm{d}C_z(u)}{\mathrm{d}u} \tag{5-23}$$

利用式（5-19）、式（5-20）、式（5-21）和式（5-23），可算出 x'、y'、z'、x''、y''、z''。再利用算出的 x'、y'、z'及式（5-17）、式（5-19）和式（5-22）可算出 u_{k+1}，此即为一阶 NURBS 逼近插值算法；同样，二阶逼近算法是将算出的 x'、y'、z'、x''、y''、z''代入式（5-18）、式（5-19）、式（5-20）和式（5-22）即可算出 u_{k+1}。这样，在每个插值周期中，参数 u 按照给定的 $V(u_k)$ 进行更新。然后利用式（5-23）算出目标位置。其中 $V(u_k)$ 可以为给定的速度值，根据加减速算得的值或其他所期望的值。将得到的 $u=u_{k+1}$分别代入 $\boldsymbol{C}(u)$ 和 $\boldsymbol{C}_{\mathrm{T}}(u)$ 就可以得到在 t_{k+1}时刻刀心点 $C(u_{k+1})$ 和相应刀轴上的点 $C_{\mathrm{T}}(u_{k+1})$。

假设刀轴单位矢量为 $\boldsymbol{T}$，则有

$$\boldsymbol{T}=[T_xT_yT_z]^{\mathrm{T}}=\frac{C_{\mathrm{T}}(u_{k+1})-C(u_{k+1})}{\left\|C_{\mathrm{T}}(u_{k+1})-C(u_{k+1})\right\|} \tag{5-24}$$

如图 5-9 所示，机床配置为工作台摆转立式 5 轴 CNC 机床。刀具随 Z 轴上下移动，摆动台和转动台绕工作台摆转支点 $\boldsymbol{P}_{\mathrm{S}}$ 旋转指定的方位（B，C）。两个旋转角 B 和 C 可由式（5-25）确定：

$$\begin{cases} B=\begin{cases}\arctan\left(\sqrt{T_x^2+T_y^2}/T_z\right),\ (T_z\neq 0)\\ 90^\circ,\ (T_z=0)\end{cases} \\ C=\begin{cases}\arctan\left|T_y/T_x\right|,\ (T_x>0,\ T_y\geqslant 0)\\ 180^\circ-\arctan\left|T_y/T_x\right|,\ (T_x<0,\ T_y\geqslant 0)\\ 180^\circ+\arctan\left|T_y/T_x\right|,\ (T_x<0,\ T_y\leqslant 0)\\ 360^\circ-\arctan\left|T_y/T_x\right|,\ (T_x>0,\ T_y\leqslant 0)\end{cases}\end{cases} \tag{5-25}$$

式（5-25）中，若 $T_x=0$，则 $\arctan\left|T_y/T_x\right|=90^\circ$。

（3）数据流的运动　从 CAD/CAM 中生成给定格式的双 NURBS 数控代码

后，经译码器处理传给插值器；插值器根据给定的插值周期算出各轴运动目标值，再由 Axis 模块加载控制规律。

由于数控系统采用了 SoftSERCANS 技术[122]（国标 IEC61491-SERCOS 技术的软件实现，如图 2-4 所示），得到的数据被放入内部报文缓冲区（一块共享内存区），然后 SoftSERCANS 插卡与内部报文缓冲区交换数据（读写报文），接着 SoftSERCANS 通过光纤环向各伺服驱动器发出包含运动指令的数据报文从而驱动电动机，整个过程如图 5-10 所示。

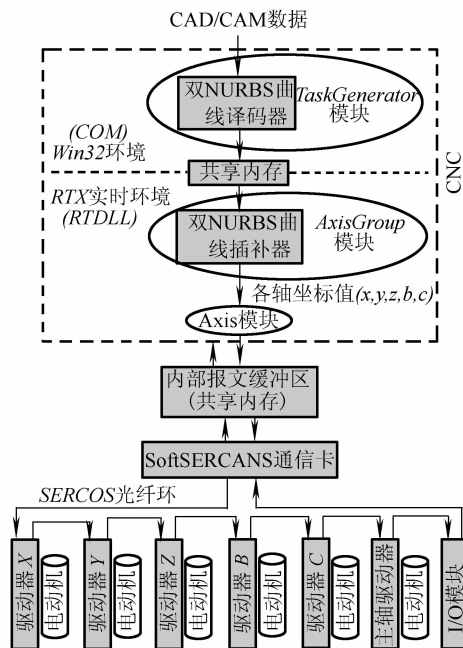


图 5-10 数据流的运动

（4）实时性能测试 本书在一台主频为 2.4GHz，内存 512M 的计算机上对给出的二阶近似五轴联动双 NURBS 插值算法进行了测试，结果如图 5-11 所示。

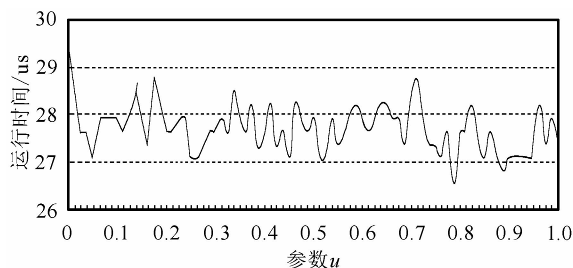


图 5-11 给出方法的实时性能测试

各时间段的起始点为计算基函数的开始，终点为算出各轴的位置。从图 5-11 中可见，在一个双 NURBS 插值周期内，平均运行时间为 28μs，该时间花销在插值周期为 2ms 甚至是 500μs 的数控系统中只占很小的一部分。

（5）加工误差分析 插值精度是评价一个插值方法好坏的重要指标。传统的线性插值程序段是在 CAM 阶段将曲线离散得到的，而本书提出的双 NURBS 样条插值方法是直接对曲线进行插值。虽然两者都以弦线逼近曲线，都存在弦高误差，但是由于本书方法逼近曲线的弦线为插值步长而线性程序段表示的线

段通常为插值步长的好几倍，因此加工表面的精度较高。

对一段刀具轨迹分别采用双 NURBS 样条插值和传统的五轴线性插值方法，弦高误差在每个插值点处的曲率圆内计算得到，经采集数据可绘制出如图 5-12 所示的误差图。图中所示的刀具轨迹曲线拐点在参数值（节点矢量）$u=0.42$ 附近，曲线曲率最小，其余地方曲率较大。从中可以看出，与传统线性段逼近曲线加工方式相比，本书的方法可以极大地改善加工性能，提高加工精度，而且加工精度受曲线曲率变化的影响较小。

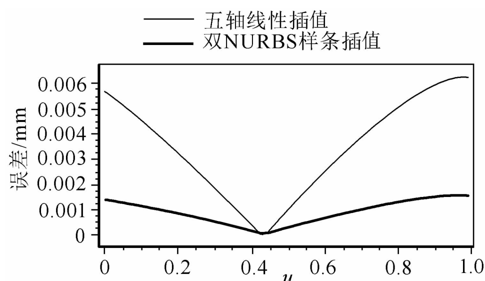


图 5-12 两种方法的误差

5.4.5 加减速处理

应该注意到，前述方法仍为弦长逼近弧长的插值方式，因此，同样会存在弦高误差 e_{k+1}，e_{k+1}应为参数 u 的函数 $e_{k+1}(u)$，可计算为

$$e_{k+1}(u)=\left\|\boldsymbol{C}_0(u)-\boldsymbol{L}_{k+1}(u)\right\| \quad u\in[u_k,u_{k+1}] \tag{5-26}$$

式中，$\boldsymbol{L}_{k+1}(u)$表示 $\boldsymbol{C}_0(u_k)$和 $\boldsymbol{C}_0(u_{k+1})$所形成线段的矢量函数表达式，即有

$$\boldsymbol{L}_{k+1}(u)=\boldsymbol{C}_0(u_k)+\frac{u-u_k}{u_{k+1}-u_k}[\boldsymbol{C}_0(u_{k+1})-\boldsymbol{C}_0(u_k)] \quad u\in[u_k,u_{k+1}] \tag{5-27}$$

式（5-26）为插值算法的理论误差模型，通过求一阶导并令其为 0 的方法可以求得最大插值误差 E_{k+1}。但这种方法计算比较复杂，考虑到 $\boldsymbol{C}_0$（u_k）和 $\boldsymbol{C}_0$（u_{k+1}）形成的弦已经为微小线段，因此可采用下述曲率圆逼近方法近似求得最大插值误差（见图 5-13）。

由图 5-13 可得

$$E_{k+1}=\rho-\frac{1}{2}\sqrt{4\rho^2-l^2} \tag{5-28}$$

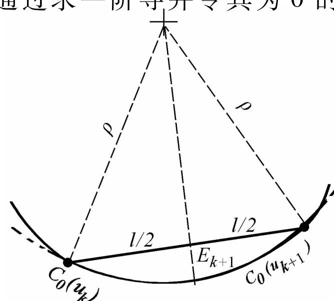


图 5-13 插值误差计算模型

式中，l 为 $\boldsymbol{C}_0$（u_k）和 $\boldsymbol{C}_0$（u_{k+1}）形成的弦长；ρ 取为 $\boldsymbol{C}_0$（u_k）和 $\boldsymbol{C}_0$（u_{k+1}）处曲率半径 ρ_k 和 ρ_{k+1}的平均值。

1. 样条段内速度自适应控制

由式（5-28）可知：①在等弦长的情况下，样条曲线曲率较大（曲率半径 ρ 较小）的地方插值误差也较大；②在曲率（或曲率半径）一定时，弦长越长对应的插值误差越大。在待插值样条曲线一定的情况下，曲线曲率已无法调整，因此，要降低插值误差就只能通过缩短弦长的方法实现。又由式（5-26）可知，较小的进给速度对应较短的弦长，如果采用参考文献［115］的等进给速度方法进行插值，则各插值周期内进给弦长近似相等，这将导致在大曲率的地方产生较大的插值误差。因此，需要对各个插值周期的进给速度进行自动调整以消除曲率大小的变化对插值误差的影响。

假定允许的轮廓控制误差为 ε，插值指令 F 指定的进给速度为 v（即沿样条曲线加工时所能达到的最高速度），则所有插值周期的插值误差 E_k 中的最大值 $E_{\max}$必须满足如下不等式

$$E_{\max} = \max_k \{E_k\} \leqslant \varepsilon$$

由于样条曲线给定的情况下，插值误差最大值 $E_{\max}$在进给速度最高（v）、曲率半径最小（$\rho_{\min}$）时取得，令式（5-28）中 $l=vT_s$，$\rho=\rho_{\min}$，则有

$$E_{\max}=\rho_{\min}-\frac{1}{2}\sqrt{4\rho_{\min}^2-(vT_s)^2}\leqslant\varepsilon$$

所以

$$\rho_{\min}\geqslant\frac{4\varepsilon^2+(vT_s)^2}{8\varepsilon}=\rho_0 \tag{5-29}$$

上式表明，要使样条插值时的插值误差满足加工要求，曲率半径 ρ 应不小于 ρ_0，若小于该值时可能导致曲率过大而损失插值精度。而当 ρ 小于 ρ_0 时，则需要对进给速度进行实时调整。调整方法如下：

$$V(u)=\sqrt{8\rho(u)\varepsilon-4\varepsilon^2}/T_s \tag{5-30}$$

2. 加减速控制原理

程序段之间过渡转接时为减少速度变化引起的加速度冲击，一般需进行加减速处理。常见的加减速类型有线性加减速、指数加减速以及 S 型加减速，本书以线性加减速为例说明。

假定系统参数指定的加（减）速度为 a，加速结束时的目标速度为 F 指令指定的速度 v，如图 5-14 所示。

进行加减速规划的关键是减速点的预测，而减速点预测的关键则是减速距离的计算。如图 5-14 所示，阴影三角形的面积 S_D 即表示减速距离。

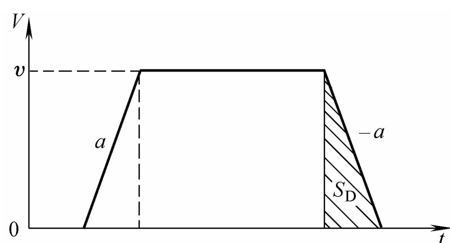


图 5-14　线性加减速模型

$$S_D = \frac{v^2}{2a} \tag{5-31}$$

需要根据 S_D 求出提前减速点的参数值 u_D。由于减速目标点为样条程序段的终点，在该点处参数值为 1，因此有

$$\int_{u_D}^{1} \left\| \frac{d\boldsymbol{C}_0(u)}{du} \right\| du = S_D \tag{5-32}$$

式（5-32）很难进行解析方法的求解。这里采用如下牛顿迭代法[136]（该方法在根的附近具有平方收敛性）来求解 u_D。

设

$$f(u) = \int_{u}^{1} \left\| \frac{d\boldsymbol{C}_0(u)}{du} \right\| du - S_D \tag{5-33}$$

则有 $f(u_D) = 0$。设 u_D 的初始值为 $u_0 = 1$，此时 $f(u_0) = -S_D$，则有如下递推表达式

$$u_{i+1} = u_i - f(u_i) / f'(u_i) \quad i \geqslant 0 \tag{5-34}$$

式中，i 取为整数。式（5-34）等价于

$$u_{i+1} = u_i + \frac{f(u_i)}{\left\| \dfrac{d\boldsymbol{C}_0(u_i)}{du} \right\|} \quad i \geqslant 0 \tag{5-35}$$

式（5-34）和式（5-35）中 $f(u_i)$ 的求取可采用复化辛普森（Simpson）积分方法[136]得到，停止迭代计算的条件为

$$\frac{|u_{i+1} - u_i|}{u_i} < \eta \quad u_i \neq 0 \tag{5-36}$$

式中，η 为迭代误差。当式（5-36）的条件成立时，表明 u_{i+1} 已足够接近 u_D，令 $u_D = u_{i+1}$ 即可。

3. 速度处理与控制的流程

通过前述分析，可将五轴 NURBS 插值过程中的进给速度前瞻控制规划如下：

1）样条程序段译码后，初始化相关数据并计算减速点对应的参数和满足精

度条件下的最小曲率半径。

2）分析当前插值点所处的区域（如加速区、减速区等），预先估算出下一插值点对应的参数并计算该点曲率半径。

3）判断计算出的曲率半径是否过小：如是则计算该曲率半径对应的进给速度，重新计算下一个插值点；如否则计算刀轴单位矢量并计算该矢量对应的旋转运动坐标。

4）判断是否到达程序段终点，如否则转至第 2）步，如是则表示到达程序段终点结束插值运算。

具体流程如图 5-15 所示。

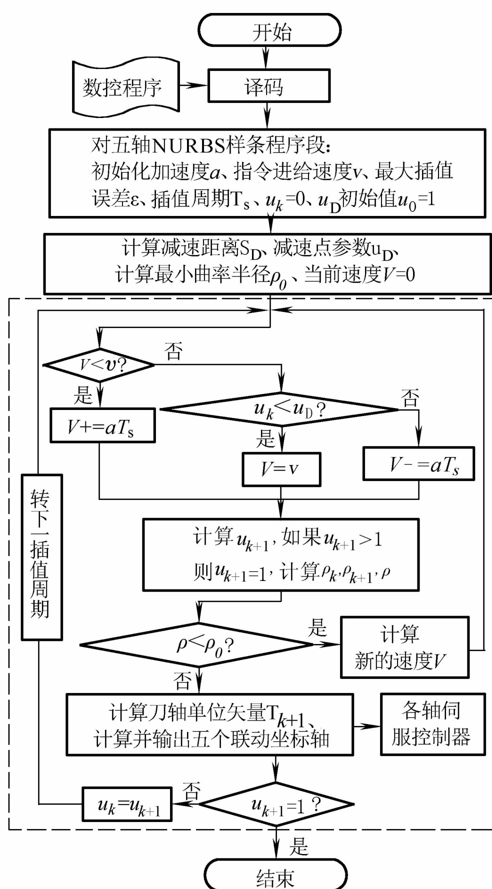


图 5-15　速度前瞻控制流程图

4. 速度控制实例计算与对比分析

假定某个待加工零件的五轴数控加工程序中包含有如下 NURBS 样条程序

段：

```
N30   G06.5 P3
X0 Y50 Z0 U0 V50 W10 R1 K0   F12000
X28.712 Y58.383 Z12.236 U29.338 V56.961 W20.758 R1 K0
X45.157 Y58.2 Z0.78 U42.235 V60.36 W11.189 R1 K0
X57.900 Y59.827 Z16.631 U61.276 V63.167 W25.335 R1 K0
X55.853 Y46.203 Z2.726 U55.158 V51.230 W10.852 R1 K0.366
X55.652 Y34.154 Z10.46 U52.334 V38.561 W19.150 R1 K0.509
X52.889 Y11.480 Z3.640 U48.352 V13.576 W12.263 R1 K0.646
X50 Y0 Z0 U50 V0 W10 R1 K0.746
K1
K1
K1
K1
```

由 F 指定的进给速度为 12000mm/min (200mm/s)。该段代码对应的刀心点轨迹如图 5-16 所示，轨迹曲线曲率最大的地方对应的曲率半径为 1.012mm。初始化加工参数如下：$a=2000\text{mm/s}^2$、$T_s=2\text{ms}$、$\varepsilon=1\mu\text{m}$、$\eta=0.001$，经译码程序计算可得 $\rho_0=20\text{mm}$，$u_D=0.926$。

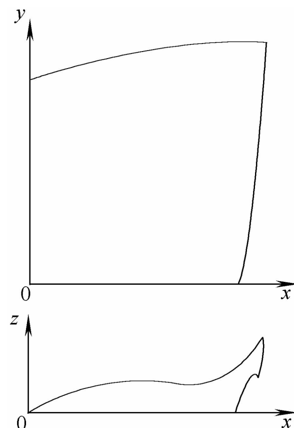


图 5-16 待插值代码对应的刀心点轨迹（单位：mm）

分别采用参考文献［115］实现的方法和本书提出的速度控制方法实时插值前面给出的五轴 NURBS 程序段，将相关数据（如插值误差、进给速度以及插值运算时间等）采集后绘制成变化趋势曲线图（见图 5-17 ~ 图 5-22）。

如图 5-17 所示，参考文献［115］的方法在曲率半径较大（如大于 ρ_0）时，插值轮廓误差还比较小，但是当曲率半径小于 ρ_0 时（尤其是在最小曲率半径处），插值误差却变得很大，达不到最终的加工要求。而本书提出的方法由于根据误差要求来计算插值点，因而可以较好地适应插值轨迹局部曲率的变化，从图 5-18 可以看出，经过速度自适应调整并修正后可以确保插值误差始终小于 ε。

图 5-19 表示的是采用参考文献［115］时进给速度的变化曲线。从该图中可以看出由于没有进行速度规划，插值输出的进给速度比较平稳，只是在曲率变化较大的地方略有波动（<5%）。本书提出方法的进给速度曲线如图 5-20 所示，在曲率半径小于 ρ_0 时，为了保证插值轮廓精度进行了适当的降速处理，而

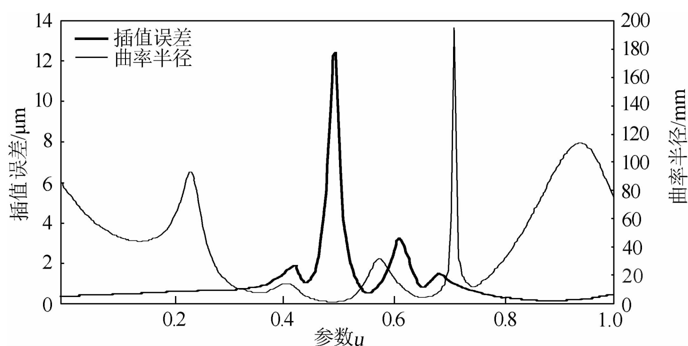


图 5-17　参考文献［115］方法的插值误差

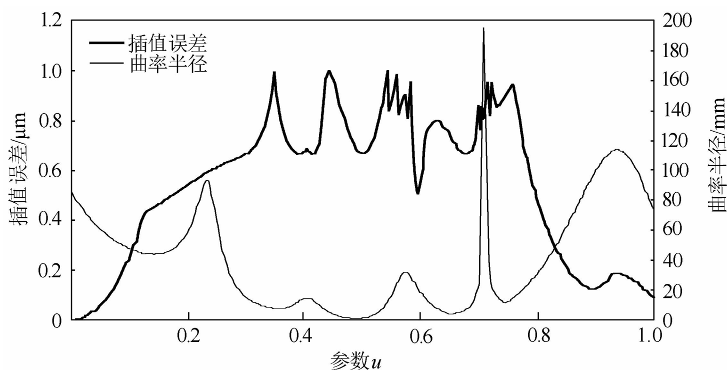


图 5-18　本书提出方法的插值误差

且在曲率最大的地方降速的幅度也最大。另外还需注意到本书的方法在程序段的起点和终点处还添加了加减速控制环节。译码程序对 u_D 的大小进行了非实时前瞻规划后，能使得程序段之间的速度过渡转接更加平稳，尽可能地减轻了机械冲击。图 5-21 所示为本书提出方法在进给速度变化时产生的加速度，加速度最大值达到了 $7\mathrm{m/s^2}$，接近 1 个重力加速度 g（$9.8\mathrm{m/s^2}$）。因此，如果机床可承受的最大加速度较小时（如 $0.5g$），还需在本书方法的基础上进一步限制加速度的大小。

图 5-22 对两种方法的运算消耗时间进行了对比。参考文献［115］的方法由于需要实时运算的任务量较小，因此插值计算速度也较快，而本书提出的方法由于添加了速度规划和处理环节后实时插值计算的任务较多，插值运算的时间有所增加（增加了 60μs 左右），但总的来看运算的时间（约 90μs 左右）对于常见系统毫秒级的插值周期而言还是足可以保证系统实时性要求的。

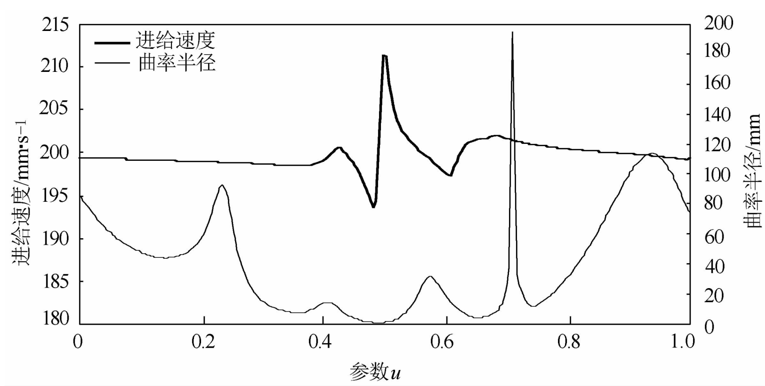


图 5-19　参考文献［115］方法的进给速度

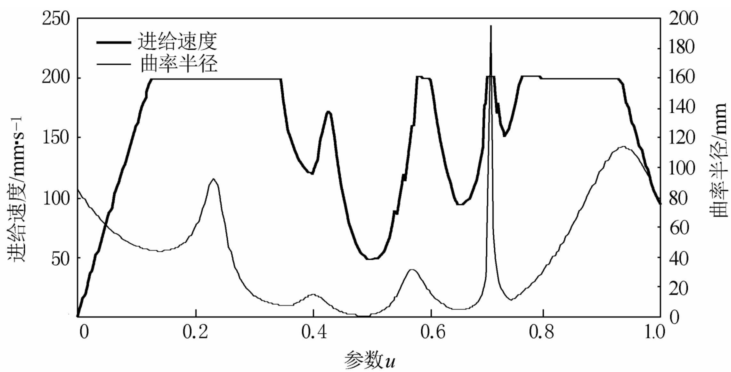


图 5-20　本书提出方法的进给速度

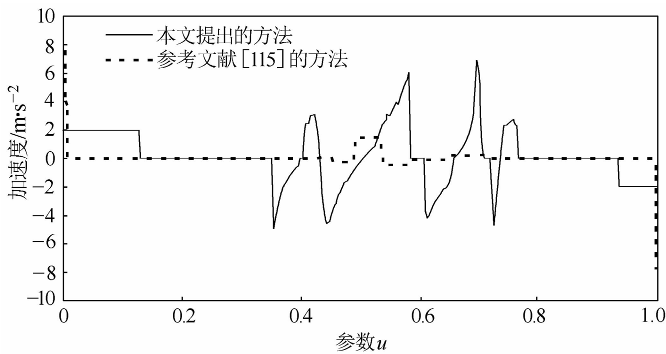


图 5-21　本书提出方法的加速度

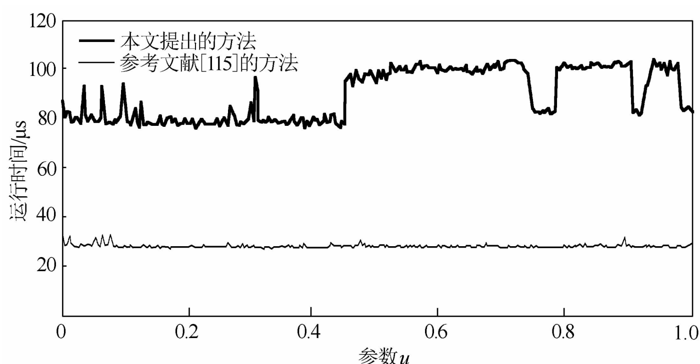


图 5-22 两种方法的实时性比较

综上可得出如下三点有意义的结论：

1）等弦长进给时插值误差随着样条曲线曲率的增大而增大，而在曲线曲率一定时，弦长越大对应的插值误差也会越大。基于此，在待插值样条曲线已经确定的情况下，曲线曲率也随之确定，减小插值误差的唯一途径是通过减小进给弦长（对应进给速度）的方法实现。

2）通过建立满足插值误差要求条件下的最小曲率半径（最大曲率）与进给速度大小之间的关系，可以准确判断当前插值点和相邻的下一插值点是否满足误差要求，并可计算满足误差要求的下一插值点。应用数值分析方法可以在插值前事先计算出样条程序段的减速点，避免了实时计算当前插值点与终点间距离以判断是否到达减速点和终点，相对减轻了实时计算任务。

3）应用本书提出的速度前瞻控制方法，即便对插值轨迹曲线上的尖角处也能有效地通过进给速度的实时调整提高插值精度并满足插值误差的要求，对五轴样条插值进行加减速速度规划使得程序段间的转接更加平稳，能有效减轻转接过程中速度变化对机床的机械冲击。

5.5 小结

本章主要研究了 Open-CNC 系统五轴联动中的实时插值算法及功能的实现方法。首先在数控系统中实现了多轴线性插值中的线性加减速控制及插值功能，也包括对其中可能出现的一些特例进行了特殊处理。然后在数控系统中实现了一种自定义格式的特殊双 NURBS 样条插值方法，该方法用于五轴联动数控系统中。在第 6 章进行的加工实验证明了本方法的正确性及可行性。

第 6 章　多轴 Open-CNC 制造系统的实现

6.1　设备联机调试及集成机制

该多轴制造系统采用 C ++ 语言开发而成，且在第 2 章中对其模块组成已有介绍，其软件平台为 Windows 操作系统 + 实时 RTX 扩展 + Visual C ++ 6.0 编译器，硬件平台为工业 PC 机 + SoftSERCANS 通信卡（PCI 总线） + SERCOS 光纤环路，而系统的调试与集成过程比较复杂。下面将以 Windows XP 系统下对五轴系统的安装调试过程进行分析与介绍。

（1）系统的硬件平台　计算机采用艾讯（AXIOMTEK）工作站，其 CPU 为 PentiumIV 3.0GHz，内存 1GB。SERCOS 通信卡为 Rexroth Indramat 公司的 SoftSERCANS PCM-S11.2 PCI 总线板卡。

X、*Y*、*Z*、*B*、*C* 及主轴伺服驱动器为 Rexroth Indramat 公司的 IndraDrive 系列的带有 SERCOS 接口的数字式驱动器。伺服电动机为 Rexroth Indramat 公司的 IndraDyn 系列交流伺服电动机。

（2）系统的软件平台　操作系统为中文版 Windows XP SP2。实时操作子系统为 RTX 6.5.1。SERCOS 通信驱动软件为 SoftSERCANS 02V06。

（3）系统调试与集成　计算机安装完 Windows XP 系统后，其默认的工作模式为 ACPI Uniprocessor，而 SoftSERCANS 的驱动程序只能在 Standard PC 模式下工作，因此首先需变更计算机的工作模式。另外，SoftSERCANS 通信卡的使用必须有 RTX 实时系统的支持，因此安装 SoftSERCANS 驱动之前必须先安装 RTX。由此，第二步应是安装 RTX 及 VC 编译器。接下来可以安装 SoftSERCANS 02V06。

现在可以将 PCM-S11.2 插入 PCI 插槽，并运行 SoftSERCANS，若出现中断冲突（SoftSERCANS 提示 Invalid interrupt），可以在硬件设备管理器中将与之冲突的设备停用，或更换 PCI 插槽。

此时运行 SoftSERCANS 时，会出现“Drive addresses not correct”等提示，因为还需要设置相关 Y、S 等参数值。这是由于不同的 SERCOS 硬件系统（SERCOS 站）对应的系统参数不同。SoftSERCANS 修改参数的界面如图 6-1a 所示。

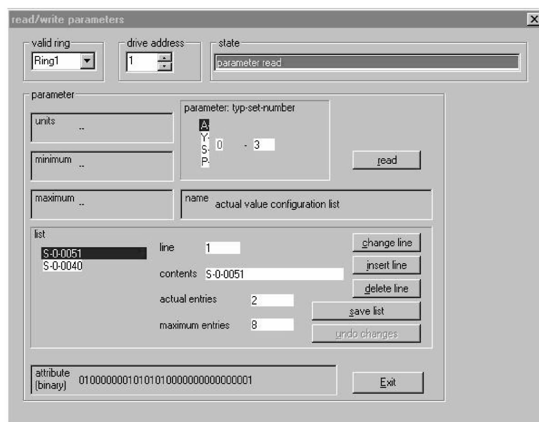
若出现以下错误，则应进行相应的设置：

1）Drive addresses not correct（驱动器地址错误）。选 service channel 中的 Y-0-0002 参数，输入驱动器或 I/O 模块站号（1、2、3、4、…）。

2）Configaration error（actual value channel）（实际值通道配置错误）。选 A-0-0003 参数，修改各驱动器（1、2、3、4、…）的参数值（进给轴驱动器修改为 S-0-0051，S-0-0040，主轴修改为 S-0-0040，I/O 站改为 S-2-0010，S-2-0030 等）。

3）Congfigration error（Command value channel）（命令值通道配置错误）。选 Service channel 中的 A-0-0002 参数，进给轴驱动器修改为 S-0-0047，主轴修改为 S-0-0036，I/O 站改为 S-2-0000，S-2-0020 等；再运行 SoftSERCANS，若显示："SoftSERCANS in operation mode"，则 SERCOS 卡彻底配置成功。下一步可进行数控系统相关动态链接库等的设置与注册了。

SoftSERCANS 的成功运行后的界面如图 6-1b 所示。



a)



b)

图 6-1 SoftSERCANS 运行界面

a）修改参数的界面 b）成功运行后的界面

接下来设置 RTX 与 SoftSERCANS 库文件。

首先将安装后的 RTX SDK 及 SoftSERCANS 目录下的 Include 及 Lib 目录中的头文件及库文件复制安装 VC6.0 的相应目录中。

然后设置相关库以注册 COM 组件，将 SOFTSERCANSEXE.DLL 复制 systems32 中。

最后注册译码模块 NISTTASKGenerator.dll 及轴组模块 AxisGroup Module.dll 即可。

运用 SoftSERCANS 不方便设置系统所有的参数。力士乐还提供了一个工具软件 IndraWorks Engineering 进行驱动器和电动机参数设置等的设置。IndraWorks Engineering 是基于 Microsoft.NET 技术开发的为所有驱动和控制系统设计的直观工程环境，该系统带有用户需要用来设计规划、参数设定、诊断和试车的所有工具，并实现一个完整性能的用户接口。其工作界面如图 6-2 所示。

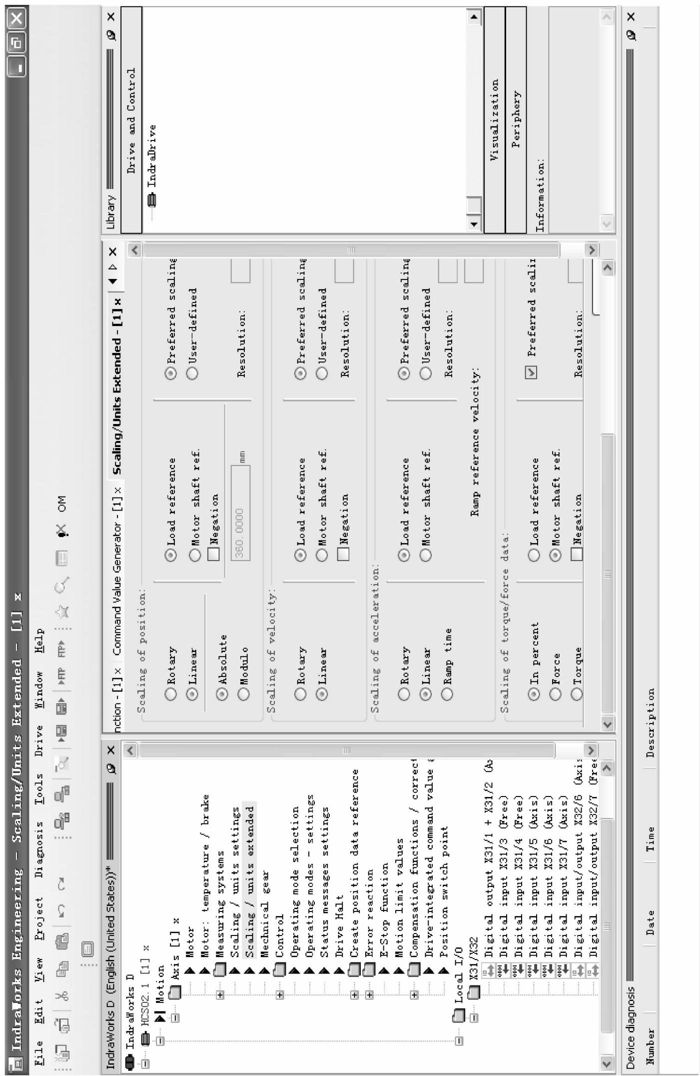


图 6-2　IndraWorks Engineering 工作界面

6.2 制造系统的搭建

6.2.1 三轴系统

系统的软硬件平台及构成如下。

(1) 系统的硬件平台 采用研华工业 PC 机，其 CPU 为 PentiumIII 800MHz，内存 256MB。SERCOS 通信卡为 Rexroth Indramat 公司的 SoftSERCANS PCM-S11.2 PCI 总线板卡。

X、*Y*、*Z* 轴伺服驱动器为 Rexroth Indramat 公司的 ECODRIVE03 系列的带有 SERCOS 接口的数字式驱动器，伺服电动机为 Rexroth Indramat 公司的 MKD 系列交流伺服电动机，主轴采用高速电主轴，机床为数控立式升降台铣床。

(2) 软件平台 操作系统为英文版 Windows NT 4.0，实时操作子系统 RTX5.1；SERCOS 通信驱动软件为 SoftSERCANS 02V06。

搭建的实验台如图 6-3 所示，数控系统采用的是我们自主研发的基于 PC 机的软件化开放式数控系统配置的三轴系统，其功能可以任意更换或扩展。

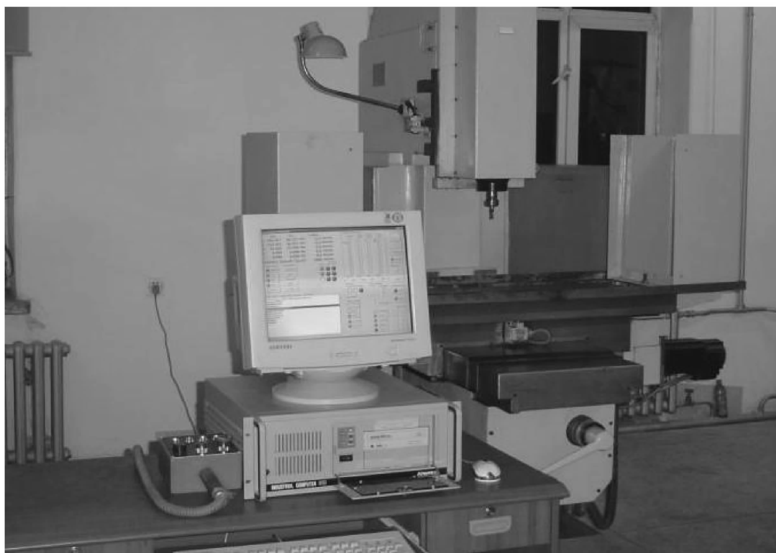


图 6-3 三轴数控铣床实验台

数控系统的人机界面（HMI）如图 6-4 所示，由于操作系统采用的是英文版，故只支持英文界面，所以数控系统界面也采用了英文。若将操作系统更换

为中文版，也可根据需要将界面配置为中文版。

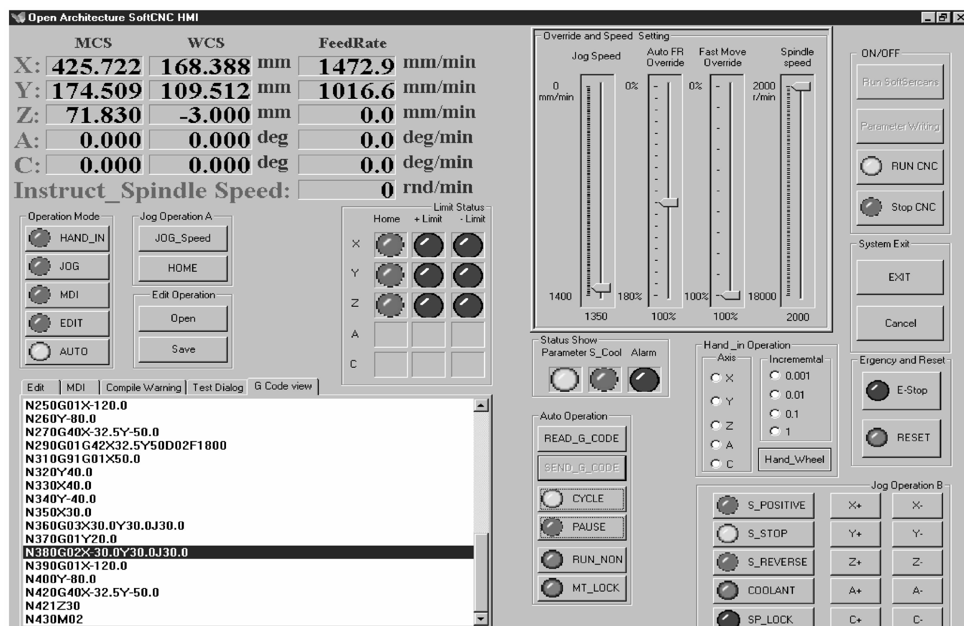


图 6-4　三轴数控系统人机界面

在该人机界面下，可以完成以下主要功能：

（1）PLC 状态显示及监控功能　显示当前 PLC 系统的各运行状态，如是否处于编辑、自动、手动状态，是否遇到限位等；实时扫描系统的运行状态，发现故障立即报警，并做出相应的处理。

（2）编辑（EDIT）功能　系统集成数控代码编辑器，可实现数控代码的输入、修改、打开、存储及译码报错等功能。

（3）MDI 功能　可以设定系统的相关参数，如参考坐标系等。

（4）数控功能显示　可以实时动态显示机床坐标系（MCS）、工件坐标系（WCS）下的坐标值、进给速度、主轴转速，并可动态显示加工零件的数控代码及突出显示当前加工的数控代码段。

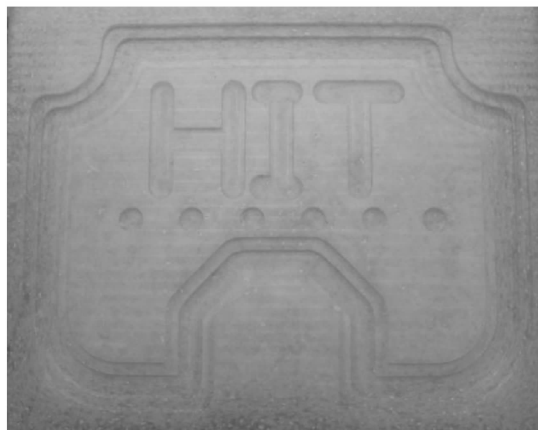
（5）自动（Auto）加工及手动（JOG）、手摇脉冲发生（Hand-In）功能　可以按照数控代码完成零件的自动加工，单独粗略或精确地控制各坐标轴或主轴的运动。并可对自动时的工进倍率、快进倍率、主轴倍率及手动时的进给速度、主轴转速进行设定。自动状态时还可暂停和中止加工。

（6）辅助功能　实现机床锁住（MT-Lock）、空运行（Run-No）、回原点（Home）等功能。

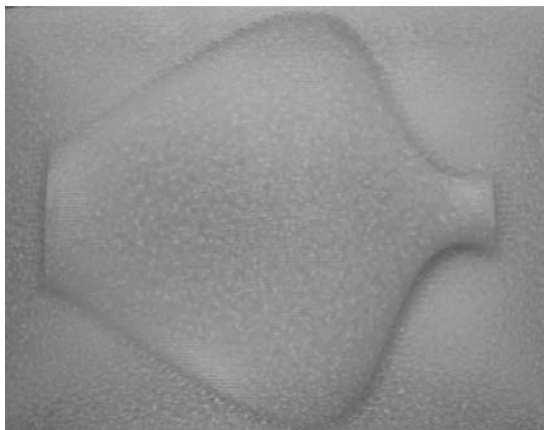
上述部分功能分别由不同的视窗提供操作界面，视窗的切换由相应的按钮实现。

本系统实现了三轴联动 CNC 的基本功能，包括常规 G、M 代码的译码（如直线（G00、G01）、圆弧（G02、G03）的译码，C 刀具半径补偿、长度补偿，坐标系选择，缩放，旋转，镜像，子程序调用，循环加工等功能）及直线和圆弧插值等功能，建立了一套实用化的数控系统。

图 6-5a 所示为该系统下采用直线、圆弧插值功能及刀具半径补偿、固定循环加工的蜡模试件。图 6-5b 所示为该系统下加工的花瓶蜡模。



a)



b)

图 6-5　三轴系统加工的蜡模

a）蜡模试件　b）花瓶蜡模

6.2.2　五轴系统

（1）系统构成　系统的软硬件平台在 6.1 节已作了介绍，这里不再赘述。

机床为 XKV715 型工作台摆转式五轴联动铣床，数控面板为江苏姜堰市赛洋数控设备有限公司的矩阵加密控制面板。机床装置如图 6-6 所示。



图 6-6　五轴机床装置

五轴数控系统的人机界面如图 6-7 所示。

（2）加工实验　进行的加工实验分几种类型分别进行。

1）利用该系统的圆弧插值功能对蜡模毛坯进行加工。如图 6-8 所示，其中图 6-8a 为加工过程，加工速度为 4000mm/min，图 6-8b 为加工好的毛坯。

2）利用五轴联动直线插值功能加工蜡模叶片。如图 6-9 所示，图 6-9a 为加工过程，加工速度为 3500mm/min，图 6-9b 为加工好的蜡模叶片。

3）利用五轴联动直线插值功能加工铝件叶片。如图 6-10 所示，其中图 6-10a 为加工过程，加工速度为 3500mm/min ，图 6-10b 为加工好的铝件叶片。

4）验证五轴联动双 NURBS 样条插值功能。通过试切叶片开槽的轨迹来验

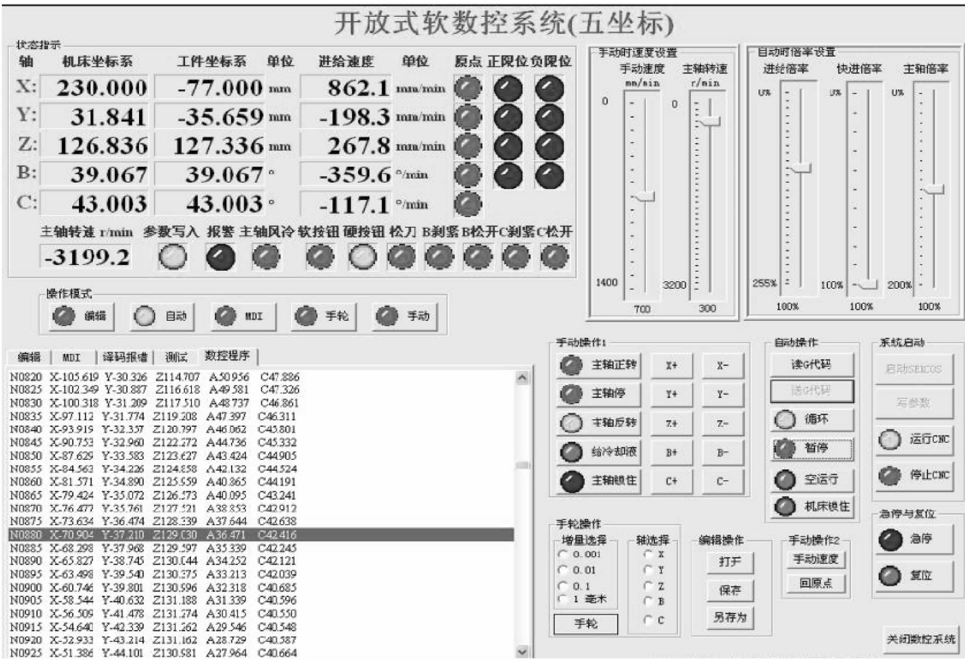
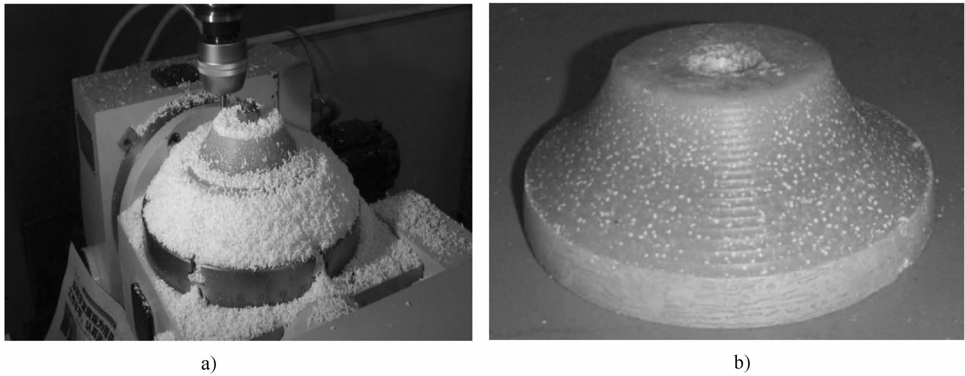


图 6-7　五轴数控系统人机界面

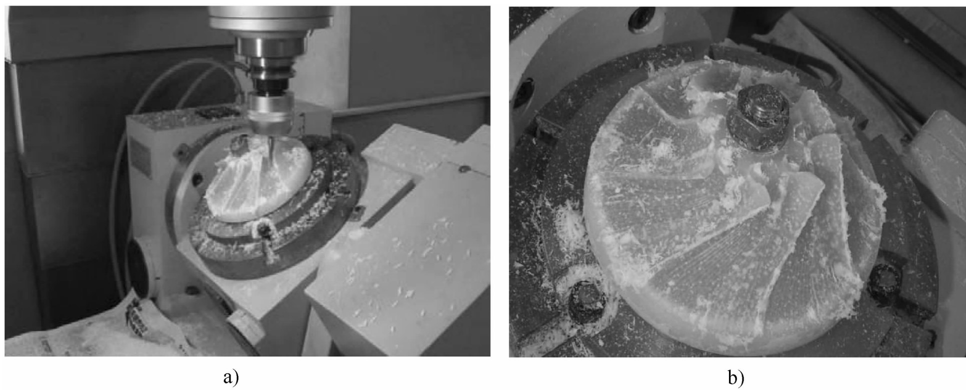


a)　　　　b)

图 6-8　蜡模毛坯加工

a）加工过程　b）加工好的毛坯

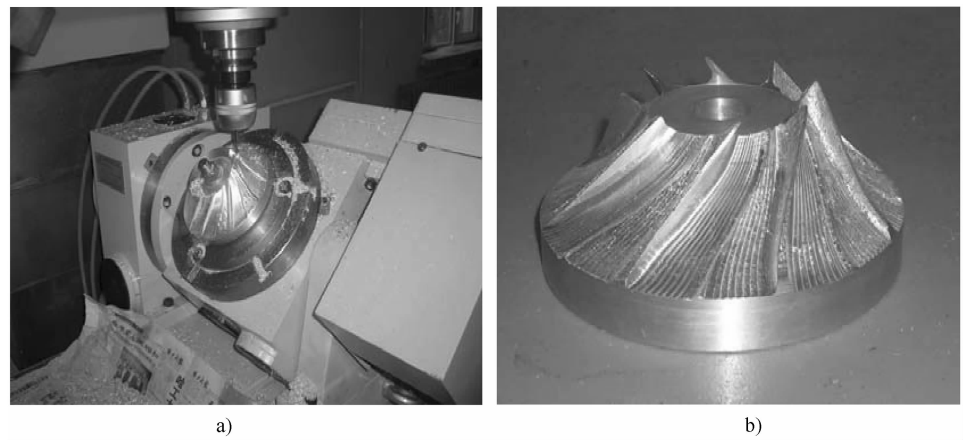
证系统中添加的五轴双 NURBS 样条插值功能。由 CAM 前处理时得到的刀位路径经后处理后分别拟合成刀心样条 C（u）和方位样条 C_T（u），如图 6-11a 所示。刀具路径如果仍然采用五轴线性插值方法，则得到的数控加工代码将由 40 余个程序段组成，如图 6-11b 所示。而如果采用本书的五轴 NURBS 样条插值方法，则用一个程序段即可替代，该段代码如图 6-12 所示。



a)　b)

图 6-9　蜡模叶片加工

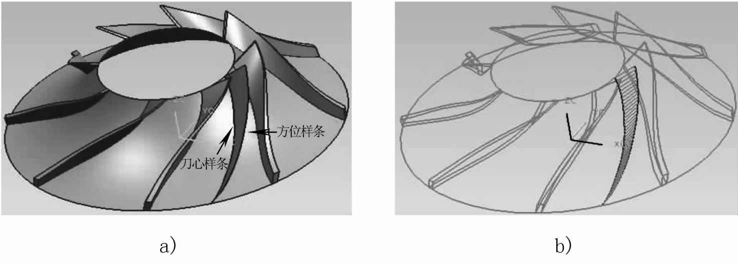
a）加工过程　b）加工好的蜡模叶片



a)　b)

图 6-10　铝件叶片加工

a）加工过程　b）加工好的铝件叶片



a）　b）

图 6-11　五轴双 NURBS 样条

a）五轴双样条路径　b）五轴线性插值路径

```
N30 G06.5 P3 F300
X46.4686 Y-58.8701 Z10.0000 U50.4292 V-55.5150 W14.9999R1.0 K0.000
X46.9729 Y-56.0110 Z10.5478 U50.6705 V-52.8277 W15.8250R1.0 K0.000
X46.9602 Y-50.3135 Z11.9364 U50.9018 V-47.4858 W17.6598R7.0 K0.000
X45.7889 Y-42.0990 Z14.7727 U50.6079 V-39.6753 W20.9482R1.0 K0.000
X44.8936 Y-33.9669 Z17.9421 U49.8060 V-32.1397 W24.7434R1.0 K0.125
X43.1750 Y-26.2031 Z21.6789 U48.6000 V-24.8993 W28.9699R7.0 K0.250
X40.5105 Y-19.1196 Z26.1378 U47.1904 V-17.8667 W33.4695R1.0 K0.375
X37.5797 Y-12.4928 Z31.0419 U45.2945 V-11.2104 W38.3614R1.0 K0.500
X34.2985 Y-6.2323 Z36.3315 U42.8755 V-5.1580 W43.7844R7.0 K0.625
X31.4151 Y-3.3427 Z40.5880 U40.9943 V-1.6241 W47.7837R1.0 K0.750
X29.8908 Y-2.5535 Z43.0006 U39.9993 V-0.1753 W50.0004R1.0 K0.875
K1.000
K1.000
K1.000
K1.000
```

图 6-12　五轴双 NURBS 代码

用第 5 章介绍的插值方法分别插值 $C(u)$ 和 $C_T(u)$ 得到插值点及各点处刀轴方位，加工的过程及结果分别如图 6-13a、b 所示。

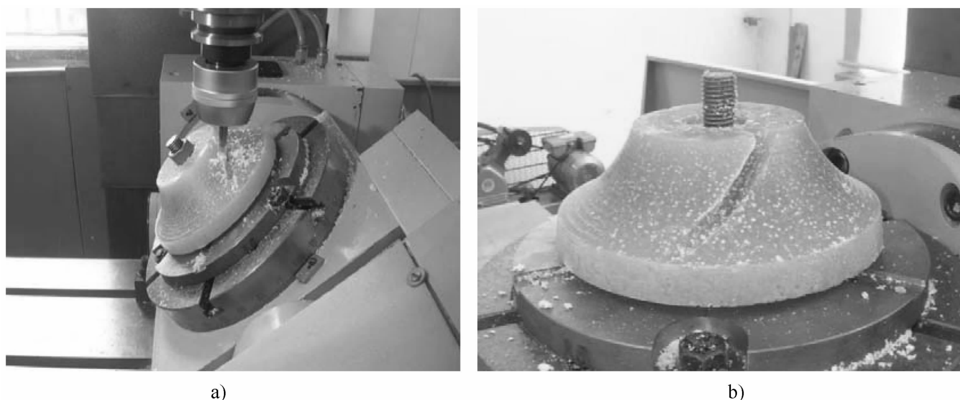


图 6-13　双 NURBS 样条加工的刀位轨迹

a）加工过程　b）加工结果

通过图 6-11 与图 6-13b 的对比可见，经过双样条插值后，机床所走的刀位轨迹与由 CAM 所设计的轨迹十分吻合，这说明所提出的双样条插值方法能很好地保证插值得到的刀轴方位与 CAM 计算的刀轴方位一致，也验证了该方法的正确性。

因本书的侧重点是开发 CNC，而不是 CAM，所以只用一段双 NURBS 样条代码对 CNC 的双样条插值功能进行了验证。采用双 NURBS 样条代码描述完整叶片并进行加工将是本课题后续五轴联动数控编程系统研究的任务，本实验室已有相关人员对此展开研究。

6.2.3　系统的开放性

系统的开放性（可互换性，可扩展性等）体现在以下几个方面。

（1）系统的硬件可以进行置换　本系统可以置换符合 SERCOS 通信标准的驱动器与电动机。例如：驱动器可由 ECODRIVE03 系列更换为 IndraDrive 系列，电动机由 MKD 系列更换为 IndraDyn 系列。本系统的计算机硬件也可以升级或更换，在不改变系统软件（包括 CNC 软件）的情况下，可更换更高主频的 CPU 及更大容量的内存，以提升 CNC 系统的性能。

（2）系统的操作系统可以更新　Windows 可由 NT 升级为 XP，RTX 实时子系统也可进行相应的升级。

（3）CNC 系统的功能可以进行扩充　例如：可以改变译码模块及轴组模块，在原系统基础上扩充至五轴 NURBS 插值方法。也可根据用户的需要，增加新的功能。系统的联动轴数也可根据用户的需要进行更改，且只需更改相应的系统模块即可。

在 6.1 节已述及对译码模块及轴组模块的注册，这里不再赘述。轴组模块是 RTX 环境下的实时动态链接库，通过注册含有不同插值算法的 RTDLL 动态链接库的方法，就可以实现插值算法以及轴数的变动。

另外，也可更改系统的人机界面，根据需要选择界面语言等。

6.3　Soft-PLC 系统实验

6.3.1　开发系统

开发系统用户界面如图 6-14 所示。

界面可分为七个部分：菜单栏、工具栏、状态栏和四个视窗（梯形图编辑视窗、指令表编辑视窗、错误显示窗口和目标代码显示窗口）。

用户在梯形图编辑视窗中编辑梯形图命令，转换为指令表后显示在指令表编辑视窗中。指令表可以进一步被编辑，编译后可能显示错误。通过错误显示窗口中提供的错误信息调试指令表，到程序不再有错误的时候，生成目标代码并显示在目标代码显示窗口中。

编辑梯形图控制命令，图 6-15 所示为五坐标数控铣床 PLC 控制部分梯形图。通过编辑模块中的转换功能，生成指令表为

……

025：LD LI0.7

026：ANI PI2.7

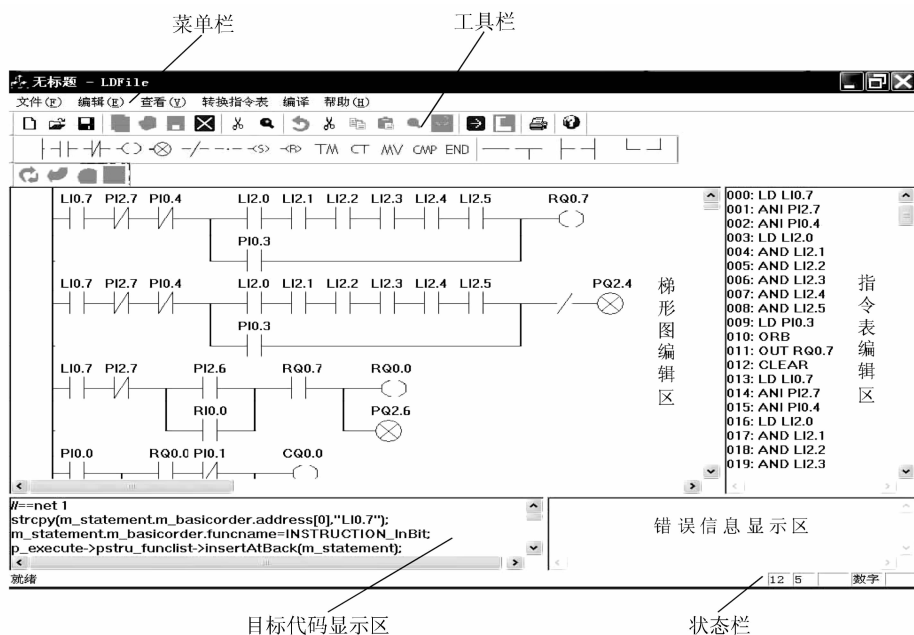


图 6-14 开发系统用户界面

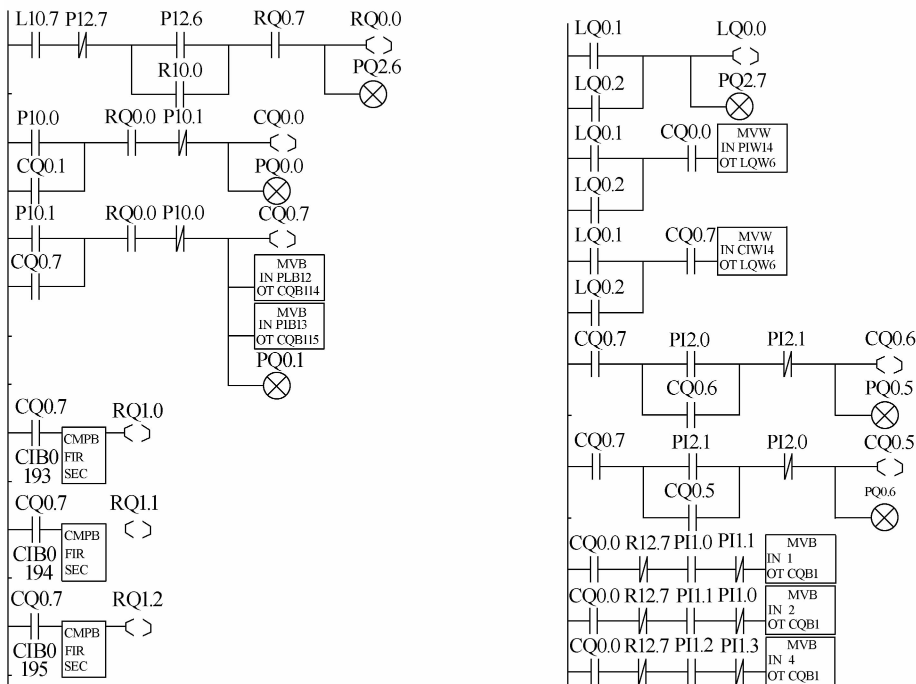


图 6-15 逻辑控制梯形图

027：LD PI2.6

028：ORB RI0.0

……

通过编译模块将生成的指令表语言进行调试、编译，生成目标代码为

```
// == net 3
strcpy (m_statement. m_basicorder. address [0], "LI0.7");
m_statement. m_basicorder. funcname = INSTRUCTION_InBit;
……
```

还可以将该目标代码处理成相应的二进制格式存为一个文档。然后将此目标代码或其对应的二进制代码传递到数控铣床的运行系统上，即可实现机床基本的控制功能。

6.3.2　运行系统

Soft-PLC 结合 CNC 构成了基于工业 PC 机、RTX 实时操作系统的完整的软件化数控系统。下面以五轴平台为例，验证 Soft-PLC 运行系统的性能。

Soft-PLC 多任务系统为软件化数控系统的一个子系统，该数控系统的多任务包括：

1）实时中断任务。通过定时中断执行运动控制，对数控程序中运动指令进行加减速、插值计算，输出位置控制指令值和辅助功能指令。

2）用户监控任务。HMI 响应用户按键输入，并将各输入指令值保存到对应的命令寄存器供 PLC 扫描，同时读取状态寄存器中的信息刷新界面显示。

3）后台任务。PLC 扫描包括 HMI、CNC 以及外部 I/O 的输入，并按照优先级执行不同任务等级的用户逻辑控制程序，输出逻辑控制结果。

数控系统按照固定的任务优先级和中断机制执行各个任务功能：实时中断任务具有最高优先级，其实时执行是数控插值控制精度的保证；后台任务是在实时中断任务各个时间片剩余的时间内执行 PLC 循环扫描任务，PLC 任务内部调度各子任务完成输入输出、通信和程序执行功能；用户监控任务是处在低优先级操作系统内核中的进程，是数控系统中优先级最低的任务，但系统通过中断 Soft-PLC 任务来响应用户输入及状态输出。以实时中断任务执行时序为参考，数控系统多任务调度时序如图 6-16 所示。

系统在运行约 300 个梯级的用户程序时，测得的 Soft-PLC 基本扫描周期如图 6-17 所示，其中图 6-17a 为无插值任务时的扫描周期波动图，约为 3ms，图 6-17b 为数控系统进行五轴联动线性插值加工时的扫描周期波动图，约为 12ms。波动范围均在 3% 以内，可见系统运行比较平稳可靠。

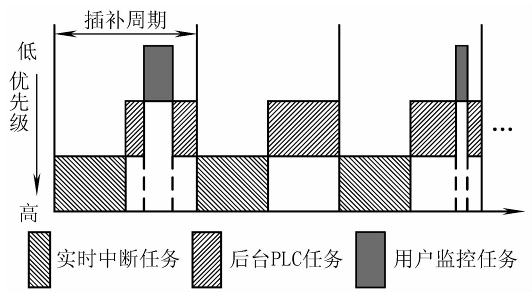


图 6-16　数控系统多任务调度时序

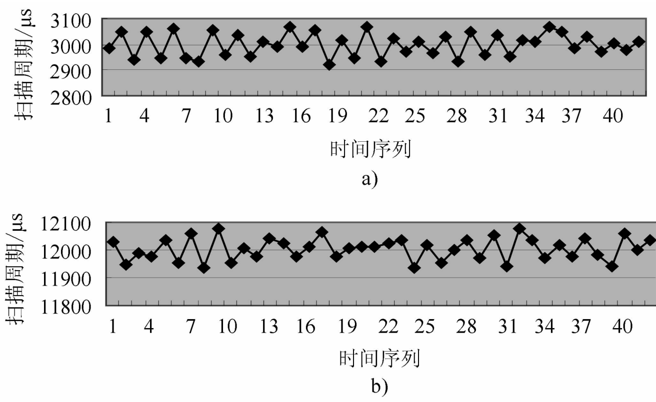


图 6-17　Soft-PLC 扫描周期

a）无插值任务　b）五轴联动线性插值加工

6.4　小结

实验研究是检验和评价系统性能的重要手段。本章给出了系统的调试方法及调试过程，并对软件化数控系统的加工能力、可靠性及开放性进行了验证，对 Soft-PLC 运行系统和开发系统的性能也进行了实例验证。

结　　论

本书在分析当前开放式数控系统研究成果的基础上，对开放式数控系统的关键技术展开了针对性的研究，并实现了一套较为完备的、与现行系统不同的、独特的、基于 IPC 机的多轴 Open-CNC 数字制造系统。

本书的主要研究工作及理论与技术方面的成果体现在以下几个方面：

1）研究并构建了一套基于 IPC 机的开放式软件化多轴数控系统平台。该平台采用 PC 机、通用的 Windows 操作系统、RTX 实时扩展子系统及遵循 SERCOS 协议的 SoftSERCANS 通信卡构建而成。能实现控制器与驱动器及 I/O 设备间的高速串行通信；实现软硬件的无关性以提高系统的开放性；实现三轴，乃至五轴联动。整个数控系统成为运行在 IPC 机上的应用程序。

2）运用 Lex 和 Yacc 编译软件，在 Visual C++的开发环境下，参考可编程序控制器国际标准 IEC61131-3，采用面向对象的 C++语言，研究开发了一套 Soft-PLC 编程开发系统，并实现了其基本功能。该系统由编辑模块和编译模块组成，编辑模块实现了梯形图编辑器、指令表编辑器及由梯形图到指令表的转换功能。编译模块由词法分析、语法分析、语义分析和目标代码生成四个部分组成，实现了将指令表语言转换为 Soft-PLC 运行系统能识别的目标代码功能。

3）为开放式多轴数控系统创建了一套 Soft-PLC 多任务运行系统。该系统参考了 IEC61131-3 软件模型的层次结构，采用面向对象技术及模块化的思想构造而成。由于采用 RTX 作为操作系统的实时扩展，整个系统的实时性得到了保障；多任务机制使得系统能充分地实现 PLC 的各项功能；良好的多任务系统调度机制及数据封装技术和模块化等手段让系统协调、可靠地工作，实验运行证明了该系统的实时性和可靠性。并解决了在单 CPU 系统资源情况下，Win32 环境中 HMI 进程与 RTX 实时环境中 Soft-PLC 并行执行的问题；提出设定最小扫描时间，定时挂起及基于 MUTEX 特性的三种解决方法。

4）构建了一套功能较为完善的开放式多轴联动数控系统，且已被成功应用到三轴及五轴联动数控铣床中。该系统具有较完整的系列化的 G、M 代码译码及多轴插值功能，还具有平面刀具半径、长度补偿，子程序调用，坐标旋转，图形缩放等功能；且界面功能完善，具备 Auto、Jog、手轮、MDI、编辑、倍率、G 代码实时动态显示、报错、报警等。系统达到实用化程度。

5）在复杂曲线插值方面，本书在数控系统中实现了一种自定义格式的

双 NURBS 样条插值方法。不同于传统的线性插值和三轴 NURBS 样条插值，运用本书的样条曲线插值方法可以进行“曲线”级的五轴联动的复杂曲面加工。

多轴 CAM/CNC 制造系统的研究包含很多方面的内容，是一个综合的、系统的研究课题，也是一个不断发展的课题。本书的研究内容相当于做了个基础和铺垫的作用，可为项目的后续研究打下坚实理论基础和前沿性探索。

参 考 文 献

[1] OSACA Work Group. OSACA Handbook Part I: Basics of OSACA [R/OL]. 1996. http://www.osaca.org/.

[2] Pritschow G, Altintas Y, Jovane F. Open Controller Architecture - Past, Present and Future [J]. CIRP Annals-Manufacturing Technology, 2001, 50 (2): 463-470

[3] GM Powertrain Group Manufacturing Engineering Controls Council. Open, Modular Architecture Controller at GM Powertrain: Technology and Implementation [R/OL]. 1996. http://www.omac.org/techdocs/open_at_GM.pdf.

[4] Wright P K. Principles of Open-Architecture Manufacturing [J]. Journal of Manufacturing Systems, 1995, 14 (3): 187-202.

[5] Bruce M A, Cole J R, Holland R G. An Open Standard for Industrial Controllers [J]. Manufacturing Review, 1993, 6 (3): 180-191.

[6] Golden E Herrin. Next Generation Controller [J]. Modern Machine Shop, 1990, 12: 148-150.

[7] Golden E Herrin. Where Is NGC Today [J]. Modern Machine Shop, 1993, 6: 142-144.

[8] Golden E Herrin. NGC Deployment [J]. Modern Machine Shop, 1993, 7: 148-150.

[9] Golden E Herrin. NGC On Its Fourth Birthday [J]. Modern Machine Shop, 1993, 8: 136-138.

[10] Jean V Owen. Opening up Controls Architecture [J]. Manufacturing Engineering, 1995, 11: 53-60.

[11] OSACA Work Group. Open System Architecture for controls within Automation System, EP 6379 & EP 9115. OSACA Ⅰ&Ⅱ Final Report [R/OL]. 1996. http://www.osaca.org/.

[12] Sperling W, Lutz P. Design Applications for an OSACA Control [C] // Proceedings of the International Mechanical Engineering Congress and Exposition. Dalles, USA, 1997 (12): 16-21.

[13] OSACA Work Group. OSACA Handbook Part II: how to Develop OSACA applications [R/OL]. 1996. http://www.osaca.org/.

[14] Sawada C, Akira O. Open Controller Architecture OSEC-II: Architecture Overview and Prototype Systems [C] // IEEE Synposium on Emerging Technologies & Factory Automation, ETFA'97. Los Angeles, USA, 1997 (9): 543-550.

[15] Herrin Golden E. Open Module Architecture Controllers (OMAC) [J]. Modern Machine Shop, 1996, 68 (11): 160-162.

[16] OMAC Architecture Working Group. OMAC Baseline Architecture Functional Requirements version1.0 [R/OL]. 2002. http://www.omac.org/.

[17] Birla S, Faulkner D, Michaloski J, et al. Reconfigurable Machine Controllers using the OMAC API [C] // Proceedings of the CIRP 1st International Conference on Reconfigurable

Manufacturing. Ann Arbor, MI, 2001 (5): 21-32.
[18] Chrysler, Ford Motor Co. , General Motors. Requirements of Open, Modular, Architecture Controllers for Applications in the Automotive Industry [R] . Chrysler, Ford Motor Co. , and General Motors. White Paper - Version 1. 1, 1994.
[19] Tan K K. Development of an integrated and Open-architecture Precision Motion Control System [J] . Control Engineering Practice, 2002, 10 (6): 757-772.
[20] Mori M, Yamazaki K, Fujishima M, et al. A Study on Development of an Open Servo System for Intelligent Control of a CNC Machine Tool [J] . Annals of the CIRP, 2001, 50 (1): 247-250.
[21] Yellowley I, Pottier P R. The Integration of Process and Geometry within an Open Architecture Machine Tool Controller [J] . Int. J. Mach. Tools Manufact, 1994, 34 (2): 277-294.
[22] Mehrabi M G, Ulsoy A G, Koren Y. Reconfigurable Manufacturing Systems: key to future manufacturing [J] . Journal of Intelligent Manufacturing, 2000, 11 (4): 403-419.
[23] Landers R G, Min B K, Koren Y. Reconfigurable Machine Tools [J] . Annals of the CIRP, 2001, 50: 269-274.
[24] Ni J. CNC Machine Accuracy Enhancement Through Real-Time Error Compensation [J] . Journal of Manufacturing Science and Engineering, 1997, 119 (11): 717-725.
[25] Kalita D. Formal Verification for Analysis and Design of Logic Controllers for Reconfigurable Manufacturing Systems [D] . Ann Arbor: The University of Michigan, 2001.
[26] Wang S, Shin K G. Constructing Reconfigurable Software for Machine Control Systems [J]. IEEE Transaction on Robotics and Automation, 2002, 18 (4): 475-485.
[27] Birla S. Software Modeling for Reconfigurable Machine Tool Controllers [D] . Ann Arbor: The University of Michigan, 1997.
[28] Moon Y M. Reconfigurable Machine Tool Design: Theory and Application [D] . Ann Arbor: The University of Michigan, 2000.
[29] Lu M C. Analysis of Acoustic Signals due to Tool Wear during Machining [D] . Ann Arbor: The University of Michigan, 2001.
[30] Robert G Landers. Supervisory Machining Control: A Design Approach Plus Force Control and Chatter Analysis Components [D] . Ann Arbor: Michigan, 1997.
[31] Moon S K. Error Prediction and Compensation of Reconfigurable Machine Tool Using Screw Kinematics [D] . Ann Arbor: The University of Michigan, 2002.
[32] Wang Shige, Shin Kang G. Reconfigurable Software for Open Architecture Controller [C] // Proceeding of the 2001 IEEE International Conference on Robotics & Automation. Seoul, Korea, 2001 (5): 4090-4095.
[33] Wang Shige, Ravishankar C V, Shin Kang G. Open Architecture Controller Software for Integration of Machine Tool Monitoring [C] // Proceeding of the 1999 IEEE International Conference on Robotics & Automation. Detroit, Michigan, USA, 1999 (5): 1152-1157.
[34] Intelligent Manufacturing Laboratory of Purdue University. Control of Machining Processes

and Systems [R/OL]. 2000. http://widget. ecn. purdue. edu/ ~ simlink/ a_ control. html.
[35] Overmars A H, Toncich D J. Application of DSP Technology to Closed-position-loop Servo Drive Systems [J]. Int. J. Adv. Manuf. Technol, 1996 (11): 27-33.
[36] Manufacturing Automation Laboratories of University of British Columbia. Intelligent Machining Module [R/OL]. 2002. http://www. mech. ubc. ca/ ~ mal/research/ ims. htm.
[37] Altintas Y, Munasinghe W K. A Hierarchical Open-Architecture CNC System for Machine Tools [J]. Annals of the CIRP, 1994, 43 (1): 349-354.
[38] Pritschow G, Lutz P. Design of a Common Data Model for Vendor-independent Open Control Systems [C] // WAC'2000 Conferrence. Idaho, USA, 2000: 458-469.
[39] Pritschow G, Daniel C, Junghans G, et al. Open System Controllers - a Challenge for the Future of Machine Tool industry [J]. CIRP Annals, 1993, 42 (1): 449-452.
[40] Mitsuishi M, Warisawa S, Hanayama R. Development of an Intelligent High-Speed Machining Center [J]. Annals of the CIRP, 2001, 50 (1): 275-280.
[41] Wang F, Wright P K. Open Architecture Controllers for Machine Tools, Part 2: A Real Time Quintic Spline Interpolator [J]. Transactions of the ASME: Journal of Manufacturing Science and Engineering, 1998, 120 (5): 425-432.
[42] Hillaire R G. New Machining Strategies with Open Architecture Controllers [D]. Berkeley: University of California, 2001.
[43] 魏仁选，陈幼平，周祖德，等．开放式数控软件的面向对象建模及其重用研究 [J]．高技术通讯，1998 (12): 30-34.
[44] 张洁，罗欣，杜润生．一种基于开放式结构的数控加工系统 [J]．机械与电子，1999 (3): 27-29.
[45] 左静，魏仁选．数控系统软件芯片的研制和开发 [J]．中国机械工程，1999，10 (4): 424-427.
[46] 范菲，陈幼平．数控系统软件芯片库集成开发环境的研究 [J]．华中理工大学学报，1998，26 (2): 11-13.
[47] 陈友东，樊锐，陈五一．基于 RT-Linux 开放式虚轴机床数控系统研究 [J]．中国机械工程，2002，13 (15): 1339-1342.
[48] 陈友东．基于组件的开放式数控系统研究与实现 [D]．北京：北京航空航天大学，2002.
[49] 陈友东，樊锐，陈五一．基于 RT-Linux 的开放式数控系统研究 [J]．中国机械工程，2002，14 (16): 1419-1422.
[50] 陈友东，陈五一，王田苗．基于组件的开放结构数控 [J]．机械工程学报，2006，42 (6): 188-192.
[51] 王敏，郇极．基于 Windows 3. X/95/NT 的开放式数控系统研究 [J]．组合机床与自动加工技术，1998 (1): 27-29.
[52] Wang Yuhan, Hu Jun, Li Ye. Study on a reconfigurable model of an open CNC kernel [J]. Journal of Materials Processing Technology, 2003 (138): 472-474.
[53] 李霞．基于 SERCOS 接口的开放式数控系统的研究 [D]．北京：北京工业大学，

2002.
[54] 郝钢，韩秋实，孙志永．基于 PMAC 的开放式数控系统性能的研究［J］．北京机械工业学院学报，2003，18（6）：24-28.
[55] 谢经明，周祖德，陈幼平，等．基于现场总线的开放式数控系统体系结构研究［J］．华中科技大学学报，2002，30（4）：1-3.
[56] 王振华，朱国力．基于现场总线的新型开放式数控系统研究［J］．中国机械工程师，2001，12（4）：395-397.
[57] 黄晓勇．开放体系结构 CNC 系统研究及研制［D］．上海：上海大学，1997.
[58] 王文，秦兴，陈子辰．基于软件构件技术的开放式数控系统研究［J］．中国机械工程，2001，12（7）：783-787.
[59] 李峰厚，叶佩青．基于 RT-Linux 的开放式数控系统研究［J］．组合机床与自动化加工技术，2001（7）：1-3.
[60] 王太勇，李波，万淑敏，等．基于现场总线的可重构数控系统的研究［J］．计算机集成制造系统，2006，12（10）：1662-1667.
[61] American Soft Servo Systems Inc. Soft motion：Software Architecture［R/OL］. 2003. http：//www. SOFTSERVO. com.
[62] 罗华丽，李斌．开放式数控系统中的 Soft-PLC 技术研究［J］．组合机床与自动化加工技术，2003（2）：38-34.
[63] 游华云，叶佩青．基于 RTLinux 的 Soft-PLC 的研究与开发［J］．计算机工程与应用，2002，38（22）：134-136.
[64] 周峰，王新华，李剑锋，等．Soft-PLC 编辑系统的设计与实现［J］．计算机工程与应用，2005，7：111-113.
[65] 王强，李彦，吉华，等．基于 RTLinux 的 Soft-PLC 研究与实现［J］．组合机床与自动化加工技术，2005（3）：56-58.
[66] 李左章，周云飞．一种基于 IPC 的内嵌式 PLC 的实现方法［J］．机械与电子，2000（5）：3-6.
[67] 高金刚，陈建春，刘雄伟．数控系统的 Soft-PLC 系统开发［J］．计算机测量与控制，2004，12（3）：254-256.
[68] G Pritschow，Y Altintas，F Jovane. Open Controller Architecture-Past，Present and Future［J］．Annals of the CIRP，2001，50（2）：463-470.
[69] 谢经明，陈幼平，周祖德，等．SERCOS 总线协议及其在数控系统中应用［J］．机械与电子，2001（5）：3-5.
[70] 梁宏斌，王永章．基于 Windows 的开放式数控系统实时问题研究［J］．计算机集成制造系统，2003，9（5）：403-405.
[71] Petrone D J，Stackhouse M D. PC-Based Control Goes Real-Time［J］．Control Engineering，1998（4）：54-64.
[72] Babb M. PC-Based Control Falls on ‘Hard’ Times［J］．Control Engineering，1997（1）：79-84.
[73] 朱文凯，王卫华，丁汉，等．基于嵌入式 PC 的开放式 Soft-PLC［J］．机械与电子，

2002 (3): 3-7.
[74] Frampton Nat, Tsao John, Yen Jerry. Hard Real-time Extensions of Windows NT Evaluation Report: Test Plan and Phase 1&2 Results [R/OL]. 1998. http: // www. omac. org / techdocs/ntrpt2. pdf.
[75] VenturCom, Inc. RTX5. 0 user's guide [R/OL]. 2002. http: //www. vci. com/.
[76] McClure C. 软件复用技术：在系统开发过程中考虑复用 [M]. 北京：机械工业出版社, 2003.
[77] Lenz M, Schmid H A, Wolf P F. Software Reuse through Building Blocks [J]. IEEE Software, 1987 (7): 34-42.
[78] 潘爱民. COM 原理与应用 [M]. 北京：清华大学出版社, 1999.
[79] 余英, 梁刚. Visual C++ 实践与提高——COM 和 COM+篇 [M]. 北京：中国铁道出版社, 2001: 195-240.
[80] 侯俊杰. 深入浅出 MFC [M]. 武汉：华中科技大学出版社, 1998.
[81] VenturCom Inc. RTX5. 1 SDK Documentation [R/OL]. 2001. http: //www. vci. com.
[82] VenturCom Inc. RTX5. 1 Runtime Documentation [R/OL]. 2001. http: // www. vci. com.
[83] Harel D. Statecharts: A Visual Formalism for Complex Systems [J]. Science of Computer Programming, 1987 (8): 231-274.
[84] Wang S, Kang K G. Reconfigurable Software for Open Architecture Controllers [C] // Proceedings of the 2001 IEEE International Conferrence on Robotics & Automation. Seoul, Korea, 2001 (5): 4090-4095.
[85] Hibbard, C Scott. SoftSERCANS-SERCOS interface Master-Connection for PC Based Control Systems [R]. Rexroth Indramat GmbH, 2001.
[86] Lutz P, Rogers G. SoftSERCANS and SERCOS interface Competency Centers [R]. ISW, University of Stuttgart, 2000.
[87] 郇极. 开放式数控系统的数字伺服接口和通信协议 [J]. 中国机械工程, 1998, 9 (5): 20-22.
[88] 杨斌, 何铁宁. 基于 SERCOS 接口的开放式数控系统研究 [J]. CAD/CAM 与制造业信息化, 2004 (6): 96~98.
[89] 陈卫福, 杨建武. 开放式数控系统及 SERCOS 接口应用技术 [M]. 北京：机械工业出版社, 2003.
[90] 王永章, 杜君文, 程国全. 数控技术 [M]. 北京：高等教育出版社, 2001.
[91] Martin H. Introduce to STEP-NC——What is STEP-NC and What Does it Cover [R/OL]. 2001. http: //www. steptools. com.
[92] 孙丽梅. 开放式数控系统中软 CNC 的研究 [D]. 北京：北京工业大学, 2003.
[93] 黄江. 开放式数控系统程序解释及刀具补偿的研究 [D]. 南宁：广西大学, 2002.
[94] 罗友兰. 刀具半径补偿在粗、精铣削中的应用 [J]. 机械制造及自动化. 2006, 2: 1-2.
[95] 李锋. C 功能半径补偿算法及实现 [J]. 机床与液压, 2005, 8: 1-2.

[96] Borges J. PC vs. PLC for Machine and Process Control [J]. Real-Time Magazine, 1997 (4): 71-72.

[97] Holman Robert. Automation and Controls Engineering, The PLC: New Technology, Greater Data Sharing [R/OL]. 2004. http: // www. softplc. com / articles. php.

[98] 黄延延，林跃，于海斌. Soft-PLC 技术研究及实现 [J]. 计算机工程，2004 (1): 165-167.

[99] Simon R. Profiles、Products& Services of PLCopen Members [EB/OL]. 1999. http: // www. plcopen. org/pages/organization/.

[100] 付应红，李晓帆，项进解. Soft-PLC (SoftPLC) 技术、产品及控制方案探讨 [J]. 微机算计信息，2000，16 (5): 27-29.

[101] Rober S J, Shin Y C. PC-Based Open Architecture PLC systems [EB/OL]. 2002. http: //www. softplc. com.

[102] Simon R. Intro IEC 61131 all parts [EB/OL]. 2000. http: //www. plcopen. org/pages/tc1_ standards/iec_ 61131_ 1/.

[103] 黄延延，林跃. Soft-PLC 技术研究及实现 [J]. 计算机工程，2004 (1): 165-167.

[104] 方原柏. PLC 发展的新动向——IEC1131-3 编程标准和开放式结构 [J]. 昆明理工大学学报，1999 (6): 102-106.

[105] 吕映芝，张素琴，蒋维杜. 编译原理 [M]. 北京：清华大学出版社，1998.

[106] 章时中. 模块化原理、设计方法及应用 [M]. 北京：中国标准出版社，1999.

[107] 林中达，王明春. 过程控制系统多任务调度技术分析 [J]. 东南大学学报，1994，11 (24): 74-77.

[108] 严爱珍. 机床数控系统用户界面的探讨 [J]. 江苏机械制造与自动化，1998 (5): 33-36.

[109] 李旗号，胡俊. 基于 Windows 95 的车床数控界面系统的研究 [J]. 机床与液压，2000 (1): 27, 41.

[110] Marchenko M, Tae J K, Lee S H, et al. NURBS interpolator for constant material removal rate in open NC machine tools [J]. International Journal of Machine Tools & Manufacture, 2004, 44: 237-245.

[111] 吴宇燕，彭志牛，王宇晗. NURBS 曲线数控插值方法及误差控制 [J]. 机械设计与研究，2006，22 (4): 75-78.

[112] 游有鹏，王珉，朱剑英. NURBS 曲线高速高精度加工的插值控制 [J]. 计算机辅助设计与图形学学报，2001，13 (10): 943-947.

[113] Yeh S S, Hsu P L. Adaptive-feedrate interpolation for parametric curves with a confined chord error [J]. Computer-Aided Design, 2002, 34: 229-237.

[114] Tsehaw Y, Ranga N. A parametric interpolator with confined chord errors, acceleration and deceleration for NC machining [J]. Computer-Aided Design, 2003, 35: 1249-1259.

[115] Qiyi, G Zhang, R Bryan Greenway. Development and implementation of a NURBS curve motion interpolator [J]. Robotics and Computer-Integrated Manufacturing, 1998, 14: 27-36.

[116] Piegl L. On NURBS：a survey [J]. IEEE Computer Graphics & Application, 1991, 11 (1) ：55-71.

[117] Lee Y S. Non-isoparametric tool path planning by machining strip evaluation for 5-axis sculptured surface machining [J]. Computer-Aided Design, 1998, 30 ：559-570.

[118] Lo C C. Real-time generation and control of cutter path for 5-axis CNC machining [J]. International Journal of Machine Tools & Manufacture, 1999, 39 ：471-488.

[119] Mahadevan B, et al. Collision-free finishing toolpaths from visibility data [J]. Computer-Aided Design, 2003, 35：359-374.

[120] 杨长祺，秦大同，石万凯. 自由曲面五轴平底刀加工路径的 NURBS 化 [J]. 重庆大学学报，2003，26（2）：82-85.

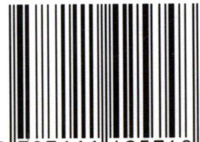
[121] Farouki R T, Tsai Y F. Exact Taylor series coefficient for variable-feedrate CNC curve interpolators [J]. Computer-Aided Design, 2001 (33)：155-165.

[122] 梁宏斌，王永章. SoftSERCANS 技术及其在数控系统中的应用 [J]. 计算机集成制造系统，2004，10（11）：1433-1436.

地址：北京市百万庄大街22号
邮政编码：100037
电话服务
社服务中心：010-88361066
销售一部：010-68326294
销售二部：010-88379649
读者购书热线：010-88379203
网络服务
教材网：http://www.cmpedu.com
机工官网：http://www.cmpbook.com
机工官博：http://weibo.com/cmp1952
封面无防伪标均为盗版

上架指导 工业技术/机械制造/数控

ISBN 978-7-111-42574-8
策划编辑◎沈红

ISBN 978-7-111-42574-8

9 787111 425748 >
定价：29.00元